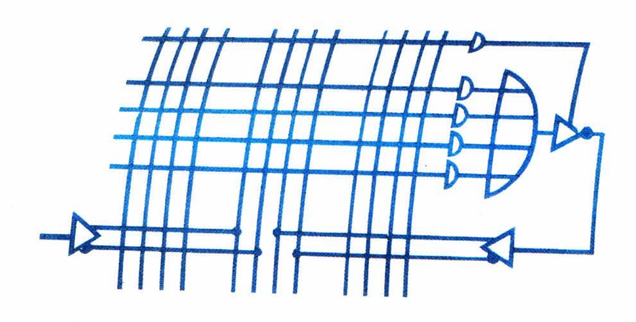
Sonderheft Nr. 208
Preis DM 28.–
öS 210.–, sfr 28.–



Programmierbare Logik

Von den Grundlagen zur Applikation





Die „dritte Welle“

Anfang dieses Jahrzehnts erschienen in den USA zwei bemerkenswerte Bücher [1, 2], deren Inhalt nur auf den ersten Blick gar nichts mit dem Thema dieses Heftes zu tun hat. Sowohl Toffler als auch Naisbitt zeigen hier, daß die Menschheit reif ist für eine Emanzipation von den Zwängen industrieller Massenfertigung und bezeichnen diesen Vorgang als „die dritte Welle“.

Herrschten einst vornehmlich agrarische Strukturen vor mit einer weitgehenden Selbstgenügsamkeit – jeder produzierte selbst, was er brauchte –, so setzte vor etwa 300 Jahren mit Macht die Industrialisierung ein. Diese „zweite Welle“ führte schließlich zur nahezu vollkommenen Trennung von Produzenten einerseits und Konsumenten andererseits. Scherze wie der von den rechteckigen Augen des „homo televisens“ haben einen durchaus realen Hintergrund – die Anpassung des Verbrauchers an fertigungstechnische Vorgaben ist eine unumgängliche Folge der Massenproduktion; Individualität ist eben teuer.

Toffler und Naisbitt beobachten nun ein zunehmendes Unbehagen an dieser unnatürlichen Abhängigkeit und prophezeien eine Kehrtwendung zu mehr Anpassung der Technik an die Bedürfnisse des Anwenders: die „dritte Welle“ rollt.
Was hat das alles mit „programmierbaren Logikbausteinen“ zu tun? Nun, auch der Elektronikentwickler war bisher gezwungen, aus den massenhaft und daher billig angebotenen Bauelementen das Beste zu machen. Er paßte sein eigenes Produkt den technologischen Möglichkeiten der Halbleiterhersteller an. So blieb es nicht aus, daß ähnliche Aufgabenstellungen zu nahzu identischen Lösungen führten; Individualität und Kreativität blieben dabei zu einem großen Teil auf der Strecke. Einen Ausweg aus diesem Dilemma bieten anwendungsorientierte Bauelemente (s. S. 3 ff.), wenn es nur gelingt, sie zu wirtschaftlich vertretbaren Preisen herzustellen. Der Mikroprozessor ist hier nicht das schlechteste Beispiel; immerhin kann man sein Verhalten allein durch Software höchst individuell bestimmen. Aber auch er stößt schließlich an – wiederum technologisch bedingte – Grenzen, die nur durch dedizierte Hardware zu überwinden sind. Und hier beginnt das Reich der „programmierbaren Logikbausteine“. Man könnte es wohl am besten so definieren:

Was mit Software nicht und mit Standard-Hardware nur mit unvertretbar hohem Aufwand realisierbar ist und darüber hinaus unverwechselbar sein soll, läßt sich mit programmierbaren Logikbausteinen ökonomisch und technisch einwandfrei umsetzen.

Einen Eindruck davon, welche ungeahnten Möglichkeiten in dieser Technik stekken, soll dieses Heft vermitteln. Wenn es dazu beiträgt, daß die „dritte Welle“ recht bald das Ufer erreicht, so hat es seine Aufgabe erfüllt.

Hans Cordes

Literatur

[1] *Toffler, A.* und *Morrow, W.:* The Third Wave, Bantam Books, New York, 1980/81. ISBN 0-553-14431-6.

[2] *Naisbitt, J.:* Megatrends, Warner Books, New York, 1982/84. ISBN 0-446-90991-2.

Inhalt

Leitartikel

Wirtschaft

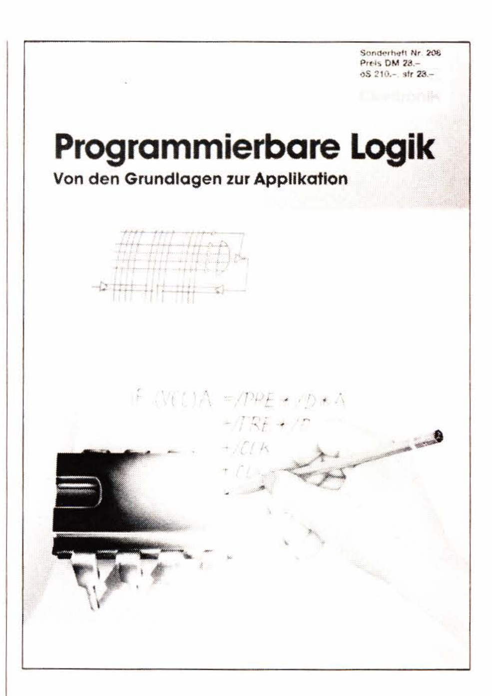
Grundlagen

Applikation

Schaltungsentwurf

Prüftechnik

Software



Zum Titelbild

Während bei RAMs und ROMs die 1-Bit-Speicherzelle und bei Gate-Arrays das NAND-Gatter mit vier Eingängen das jeweils kleinste Grundelement darstellt, wurde für programmierbare Logikbausteine der Begriff SMAC (State Machine Atomic Cell) geprägt. Diese UND-ODER-Matrix mit nachgeschaltetem D-Flipflop ist die Bewertungsbasis für die Größe und Komplexität des Bauelementes. Der Anwender muß sich jedoch bei der Enwicklung einer Schaltung kaum um das Innenleben des PALs kümmern. Software-Pakete wie PALASM erzeugen aus in einer einfachen und schnell erlernbaren Form eingegebenen Boole'schen Gleichungen die Programmierinformationen sowie Simulations- und Testdaten.

(Bild: Volker Hilbel)

Impressum

1985
Franzis-Verlag GmbH, Karlstraße 37–41, 8000 München 2.
Bearbeitet von der Redaktion der Zeitschrift ELEKTRONIK.
Für den Text verantwortlich: Hans F. Cordes

ISSN 0170-0898
Druck: Franzis-Druck GmbH, München.
Printed in Germany. Imprimé en Allemagne.
ZV-Artikel-Nr. 208021 · F/ZV/985/0049/8'

Hans Cordes

ASICs – Markt und Chancen

Auch wenn dieses Sonderheft ausschließlich den PALs (Programmable Array Logic) und ihren Derivaten gewidmet ist, so lohnt sich doch ein Blick über den Zaun in die nähere Umgebung. Schließlich stellen PALs nur einen – wenn auch bei einer ganzen Reihe Applikationen besonders günstigen – Lösungsansatz für die Realisierung sogenannter ASICs (Application Specific ICs, also „Anwendungsbezogene Integrierte Schaltungen") dar. Übrigens bedeutet die – recht häufig auftauchende – Bezeichnung „kundenspezifisch" in etwa das gleiche, erscheint jedoch semantisch etwas schwammig.

Eine wesentliche Antriebsfeder, ganze Baugruppen in einer einzigen integrierten Schaltung unterzubringen, war wohl der Wunsch des Herstellers nach Geheimhaltung seiner originären Entwicklungsideen. Ein ganz typisches Beispiel hierfür ist das geheimnisumwitterte Super-IC im Spielcomputer ZX-80 von Sinclair; ein „Knacken" seiner inneren Struktur würde einem Konkurrenten derart viel Zeit und damit Kosten verursachen, daß es noch keiner ernsthaft versucht hat. Gerade die Konsumelektronik ist wegen der dahintersteckenden Milliardenumsätze auf solche Verschleierungstaktiken – und mögen sie auch nur einige Wochen Vorsprung bedeuten – angewiesen.

Nicht weniger anregend wirkt sich die Forderung der Endabnehmer eines Produktes nach Verringerung von Volumen und Stromverbrauch aus. Voluminöse Gestellaufbauten, die noch vor wenigen Jahren das Ambiente eines „gut ausgestatteten" Labors prägten, sind längst durch handliche Geräte und Systeme abgelöst, die außerdem sehr viel leistungsfähiger als ihre Vorgänger sind. Auch hierzu ein Beispiel: Im Modell „Macintosh" von Apple ersetzen sechs PALs etwa 30 konventionelle MSI-/LSI-Schaltungen.

Zu den ASICs zählen heute folgende Gruppen an Bauelementen:

- Gate Arrays
- Standardzellen
- PALs u. ä.
- Programmierbare Festwertspeicher (PROMs usw.)
- Anwendungsspezifische Mikroprozessoren
- „Voll-kundenspezifische" Bauelemente.

Diese Reihenfolge spiegelt in etwa auch den Ausnutzungsgrad der Halbleiterfläche wider; er ist am niedrigsten bei den Gate Arrays. Die höchste Konzentration an

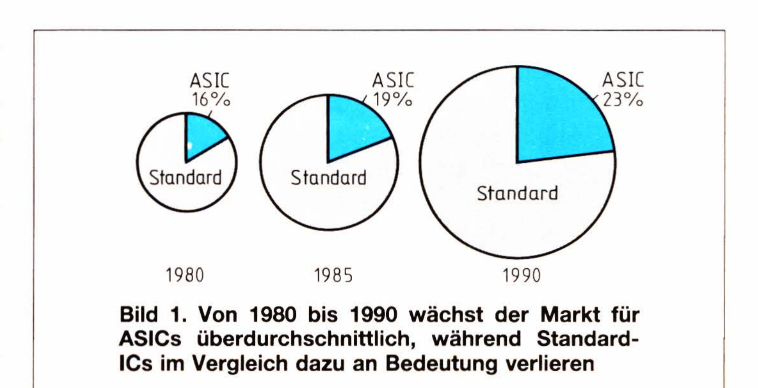


Bild 1. Von 1980 bis 1990 wächst der Markt für ASICs überdurchschnittlich, während Standard-ICs im Vergleich dazu an Bedeutung verlieren

Funktionen bei gegebener Chip-Fläche bietet der „Kundenschaltkreis", der allerdings erst bei sehr großen Stückzahlen wirtschaftlich wird.

Jede der genannten ASIC-Gruppen findet ihren speziellen Anwendungsbereich, wobei Überschneidungen durchaus keine Seltenheit sind. Die Entscheidung für die eine oder andere Lösung ist von Fall zu Fall zu fällen; eine allgemeingültige Formel kann hierfür nicht aufgestellt werden. Dagegen lassen sich aus einer 1984 von Dataquest angestellten Studie sehr wohl Aussagen ableiten, die für den gesamten ASIC-Markt gelten.

Die Grafik in *Bild 1* vermittelt einmal einen Eindruck vom wachsenden Volumen des Halbleitermarktes insgesamt, andererseits zeigt sie den steigenden Anteil der ASICs von 1980 bis 1990. Demnach verlieren Standard-ICs immer mehr an Boden; der ASIC-Markt wächst überproportional.

Eine momentan noch typische Erscheinung beim ASIC-Markt ist die überwiegende Verwendung solcher Bauelemente im eigenen Haus des Herstellers. Die Amerikaner sprechen hier vom sogenannten „Captive Mar-

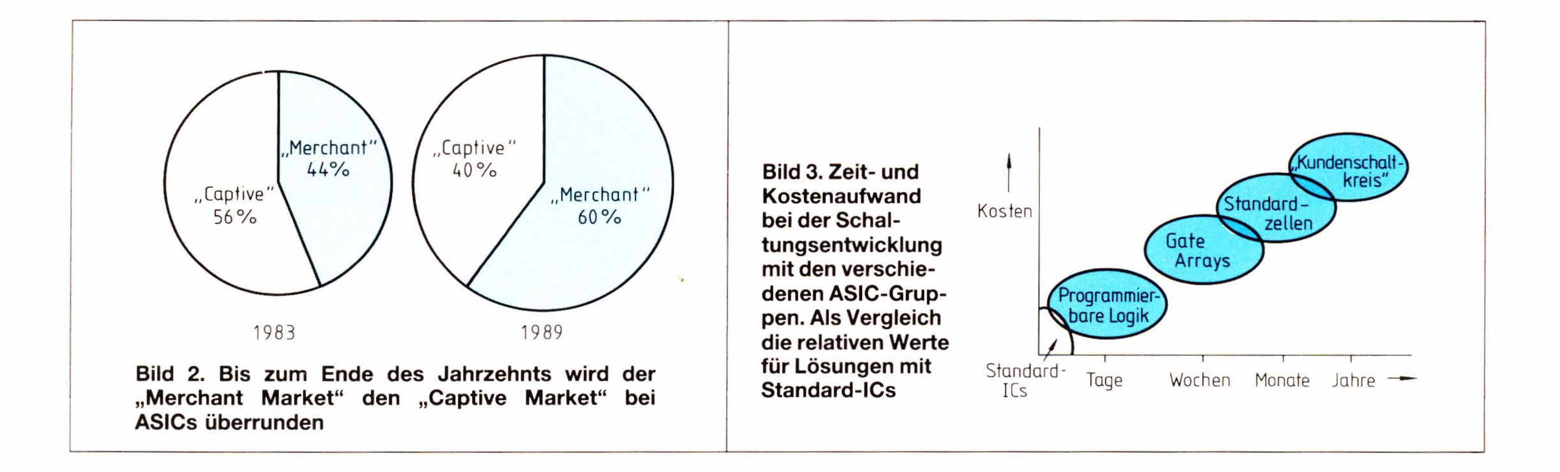


Bild 2. Bis zum Ende des Jahrzehnts wird der „Merchant Market" den „Captive Market" bei ASICs überrunden

Bild 3. Zeit- und Kostenaufwand bei der Schaltungsentwicklung mit den verschiedenen ASIC-Gruppen. Als Vergleich die relativen Werte für Lösungen mit Standard-ICs

ket". Im Gegensatz dazu steht der „Merchant Market"; er umfaßt Produkte, die im freien Handel – auch an „Außenstehende" – verkauft werden. Dataquest sieht hier bis zum Ende des Jahrzehnts eine deutliche Umkehr der Verhältnisse (*Bild 2*).

Diese Trendwende bedeutet eine größere Umwälzung als auf den ersten Blick sichtbar. So ist es durchaus nicht utopisch anzunehmen, daß ASIC-Produzenten den Herstellern von Standard-Logikbausteinen die ersten Plätze in der Weltrangliste streitig machen werden.

Mag dies momentan noch Spekulation sein, so ist eines bereits sicher: Die Distribution applikationsspezifischer ICs erfordert völlig neue Methoden. Wer hier in Zukunft bestehen will, muß dem Endanwender einen Service bieten, der weit über ein gut sortiertes Lager hinausgeht.

Hier eröffnen sich gerade für wendige Newcomer große Chancen. Besonders die programmierbaren Logikbausteine (PALs usw.) sind prädestiniert für den Vertrieb durch Handelshäuser mit Programmierservice. Die Typenvielfalt ist noch relativ gut überschaubar, womit sich die Lagerhaltung einfach und billig gestaltet. Die Anfangsinvestition ist – im Vergleich zum bisher üblichen Halbleitervertrieb – bescheiden. Was man trotz der kleinen Auswahl an Typen doch alles damit anstellen kann, zeigen ausschnittsweise die Anwendungsbeispiele in diesem Heft.

In diesem Zusammenhang ist auch *Bild 3* interessant. Es zeigt schematisch die Kosten und den Zeitaufwand bei der Entwicklung kundenspezifischer Schaltungen für die verschiedenen Gruppen von ASICs. Deutlich ist zu erkennen, daß z. B. PALs kaum schlechter dastehen als Standard-ICs – der geringe Preisnachteil wird aber oft mehr als kompensiert durch die von Haus aus gegebene Flexibilität. Auch sollte man nicht übersehen, daß sich mit solchen programmierbaren Bausteinen eventuelle Engpässe bei der Lieferung von Standard-ICs überbrücken lassen.

Om P. Agrawal, David A. Laws

Kundenspezifische Schaltungen

Welche Rolle spielen PALs

Im Bemühen, Systeme schneller, preiswerter und leistungsfähiger zu produzieren, wobei die Geräte immer kleiner werden, wählen viele Entwickler Lösungen mit Halb-Standard-Bauelementen (Semi-Custom-ICs). Definiert sind diese Bausteine als Elemente, die intern teilweise oder vollständig unverdrahtet fabriziert sind. Die endgültige Logikfunktion wird dann entweder elektrisch (mit Hilfe von Durchschmelzverbindungen) oder physisch (mit Metallmasken) entsprechend der gewünschten Funktion programmiert. Als die zwei wichtigsten Vertreter dieser Bausteinklasse haben sich heute Gate Arrays und prorammierbare Logikbausteine (Programmable Logic Devices – „PLDs") herauskristallisiert. Sie sind seit etwa zehn Jahren erhältlich. Nach einem langen Lern- und Akzeptanz-Zyklus stellen beide Techniken heute wachsende Marktsegmente dar. Während bei Gate Arrays die Weiterentwicklung in Richtung höherer Schaltungsdichten ging, legte man bei den PLDs mehr Wert auf Optimierung der Funktionen für Steuer- und Interface-Anwendungen, bei denen hohe Schaltungsdichte weniger wichtig ist als architektonische Flexibilität. Als Resultat stehen dem Systementwickler heute zwei leistungsfähige Halb-Standard-Techniken zur Verfügung, die sich gegenseitig bei Aufgaben der Implementierung von Hochleistungsmaschinen der nächsten Generation ergänzen. Nachdem auf die Eigenschaften von Gate Arrays bereits umfangreich eingegangen wurde, soll dieser Beitrag die wichtigsten Merkmale von PLDs zeigen und sie in das Gesamtspektrum der vom Kunden spezifizierbaren Bauelemente einordnen.

Architektur programmierbarer Logik

Die übliche UND/ODER-Struktur eines programmierbaren Logik-Arrays (PLA) ist das Herz aller derzeit üblichen prorammierbaren Logikbausteine (*Bild 1*). Diese Struktur läßt sich zur Implementierung der Boole'schen Summen von Produkten verwenden. Weil jede logische Gleichung auf diese Form zurückzuführen ist, stellt diese Konfiguration einen universellen Baustein dar, der auch in vielen VLSI-Schaltungen Verwendung findet. Wenn UND- sowie die ODER-Matrizen jeweils mit elektrisch programmierbaren Verbindungselementen zusammengeschaltet sind, handelt es sich bei diesem Bauelement um ein „Field Programmable Logic Array" (FPLA).

Einige FPLA-Typen bieten lediglich die Programmierbarkeit der UND-Matrix und besitzen ein festes ODER-Array. Andere arbeiten mit einer programmierbaren ODER-Matrix und einem festen UND-Array. Bei diesen handelt es sich im ersten Fall um PAL-Bausteine (*Bild 2*), im anderen um PROMs (*Bild 3*). Jede Konfiguration bietet spezielle Vorteile in bezug auf Wirtschaftlichkeit, Leistung und Flexibilität.

Die ersten programmierbaren Elemente, die auf den Markt kamen, waren PROMs. Diese programmierbare Version des Nur-Lese-Speichers wurde entwickelt, um das Problem der häufig zu ändernden Codierung von mikroprogrammierbaren Maschinen zu lösen. Wichtigstes Merkmal der PROM-Architektur ist, daß die Eingänge mit Hilfe eines festen UND-Arrays vollständig decodiert sind. Dieses Array besteht aus einem festen Adreßdecoder und einem programmierbaren Daten-Matrix-Speicherarray. Ein PROM mit n Eingangsleitungen und m Ausgangsleitungen enthält ein Array von 2^n Wörtern. Jedes Wort ist dabei m Bit breit. Die Eingangsleitungen dienen zur Auswahl eines der 2^n Worte. Wenn ein Muster von Nullen und Einsen am Eingang anliegt, ist exakt eine von 2^n Möglichkeiten ausgewählt. Das unter der entsprechenden Adresse gespeicherte Bitmuster wird daraufhin zum Ausgang des Bausteins weitergeleitet. Die PROM-Konfiguration ist sehr hilfreich beim Aufbau einfacher Logikschaltungen, z. B. Speicheradreßdecoder. Allerdings begrenzt das festgelegte UND-Array den Anwendungsbereich auf weniger komplexe Aufgaben. Der Grund liegt darin, daß mehrere Adressen nicht für das gleiche Wort verwendbar sind, so daß der

Bild 1. FPLA-Struktur in einem konventionellen Logikdiagramm dargestellt, a) FPLA-Array-Struktur, b) programmierbare UND-Logik und konventionelle Schaltung, c) programmierbare ODER-Logik und konventionelle Schaltung

Baustein lediglich ein Wort zu einer Zeit adressieren kann.

Feldprogrammierbare Versionen der PLAs waren die ersten Produkte, die speziell für Logikanwendungen konzipiert wurden. Ähnlich wie ein PROM kann ein FPLA mit n Eingängen und m Ausgängen m Funktionen von n Variablen realisieren. Ein Baustein besteht aus einem Adreßdecoder (UND-Array) sowie einer Datenmatrix (ODER-Array). Allerdings lassen sich beide Matrizen in einem FPLA programmieren. Dies ermöglicht es jedem UND-Gatter, auch „Don't-Care"-Eingänge zu besitzen. Daher lassen sich verschiedene Eingänge oder Adressen zur Anwahl des gleichen Wortes verwenden. Weil mehrere UND-Funktionen gleichzeitig auszuführen sind, lassen sich auch mehrere Worte im ODER-Array gleichzeitig anwählen. Damit ist es auch möglich, daß man eine Funktion in mehrere FPLAs aufteilt.

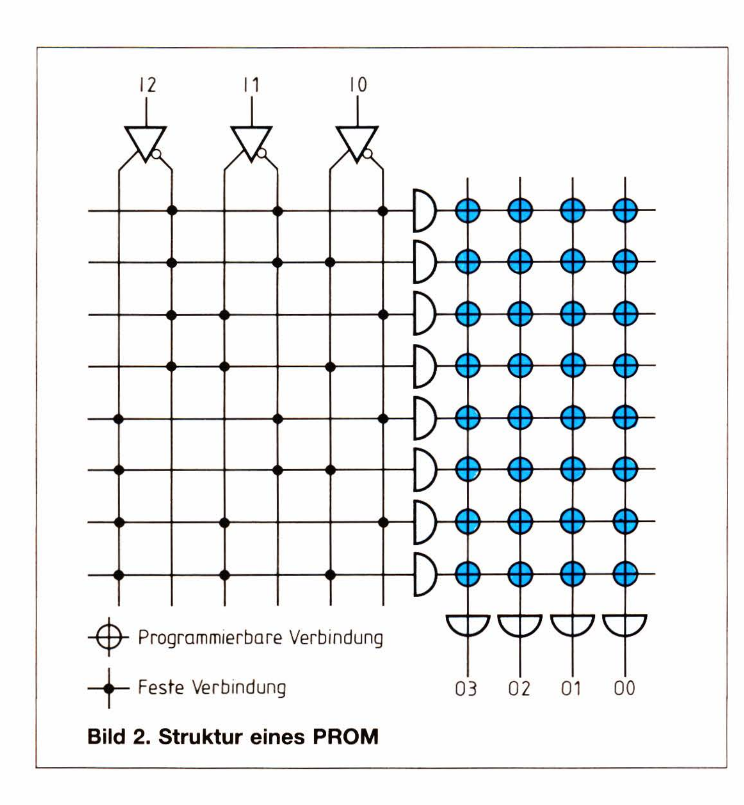


Bild 2. Struktur eines PROM

Ende der 70er Jahre erkannten *John Birkner* und *H. T. Chua*, daß nur wenige Systemanwendungen ein FPLA benötigen, bei dem die UND-Gatter speziellen Ausgangsleitungen zugeordnet sind. Dies führte zur Entwicklung der einfacheren, aber trotzdem sehr flexiblen PAL-Struktur (Programmable Array Logic). Die Architektur der PAL-Bausteine ist die spiegelverkehrte Darstellung eines PROMs. Er besitzt ein programmierbares UND-Array, aber ein festverdrahtetes ODER-Array. Damit ist es möglich, daß mehrere Adressen das gleiche Wort und mehrere Worte im Array gleichzeitig angewählt werden können. Wie FPLAs unterscheiden sich PALs von PROMs dadurch, daß sie so viele Eingänge besitzen können, wie das für eine spezielle Implementierung notwendig ist. Weil allerdings die Durchschmelzverbindung sowie die Programmier- und Test-Schaltungen im ODER-Array des PAL-Bausteines entfallen, ist eine praktische Lösung bei gleicher Technologie

sowie lithografischer Auflösung kleiner als die mit einem FPLA. Auch die Tatsache, daß weniger Schaltungsteile vorhanden sein müssen und damit auch schädliche Kapazitäten entfallen, führt zur Verbesserung der Arbeitsgeschwindigkeit, die etwa 15...20% höher liegt als vergleichbare FPLA-Typen. Einfaches Konzept und höhere Arbeitsgeschwindigkeit sorgten dafür, daß sich dieser Bauelementetyp auf dem Markt durchsetzte, so daß heute mehr als 70% der PLD-Umsätze von PAL-Bausteinen abgedeckt werden.

Konfiguration von PLD-Typen

Die drei grundlegenden Architekturen für PLDs lassen sich mit zusätzlichen Logikfunktionen erweitern. Es handelt sich dabei um Register, Latches, Rückkopplungspfade und bidirektionale E/A-Anschlüsse. Viele PLD-Konfigurationen wurden bereits mit mehreren dieser Merkmale entwickelt. Dazu zählen PROMs mit Register-Ausgängen, FPLAs mit untergeordneten Zustandsregistern und PAL-Bausteine mit hochflexiblen E/A-Ports.

Besonders interessant sind Bausteine mit programmierbaren bidirektionalen E/A-Anschlüssen. Diese sind so gestaltet, daß sie sowohl als Ein- als auch als Ausgang dienen können. Das Mischungsverhältnis läßt sich für eine bestimmte Anwendung programmieren, und der Entwickler ist nicht auf eine feste Anzahl von Eingängen oder Ausgängen beschränkt. Der Baustein AmPAL22V10 (*Bild 4*) verfügt über zehn bidirektionale und zwölf als Eingänge festgelegte Ports. Auf diese Weise kann der Baustein maximal 22 Eingänge und 10 Ausgänge haben; die Gesamtzahl ist nicht begrenzt durch die 22 Logik-Anschlüsse des 24poligen Gehäuses. Frühere PAL-Bausteine besaßen typischerweise bis zu sechs bidirektionale, zehn Ein- und zwei Ausgänge.

Die bidirektionalen E/A-Anschlüsse bieten weiterhin die Möglichkeit, den Ausgangspuffer über ein Produkt-Term zu programmieren und den Ausgang als speziellen Ausgangs-, Eingangs- oder dynamisch umschaltbaren Ein-/Ausgangs-Anschluß zu konfigurieren. *Bild 5a* stellt dies für einen 20poligen PAL-Baustein vom Typ 16L8 vor. Bei einer Ausgangskonfiguration ist der Ausgangspuffer dauernd freigegeben (alle Durchschmelzverbindungen für diesen Produkt-Term sind unterbrochen). Daher verhalten sich die Anschlüsse wie Ausgänge. Die Logikfunktion wird auf das UND-Array zurückgeführt. Um einen Eingang zu erhalten, wird das Produkt-Term nicht benutzt, so daß im Ursprungszustand der Anschluß als Eingang konfiguriert ist.

Dynamisch steuerbare Eingangs-/Ausgangs-Puffer (Sperren und Freigabe des Produkt-Terms durch logische Kombination eines oder mehrerer Eingänge) sind sehr hilfreich in busorientierten Mikroprozessorapplikationen, z. B. Datensteuerung, Datenspeicherung, Manipulation oder Anwendungen mit Handshaking-Protokollen. Diese Einrichtung ermöglicht es einem Anschluß, als Eingang (z. B. zum Schieben nach links) oder als Ausgang (z. B. zum Schieben nach rechts) verwendet zu werden. Bidirektionale Ein-/Ausgabe ist sehr hilfreich bei allen Schiebe- und Rotations-Funktionen sowie für die programmierbare Zuweisung von Eingangs-/Ausgangs-Anschlüssen.

Der Grad an Programmierbarkeit und Komplexität eines PLD wird von der Zahl der Durchschmelzverbindungen festgelegt, die den Programmierteil der UND- und ODER-Gatter darstellen. Diese können mit Emitter-Folgern oder Dioden-Matrix-Schaltungen aufgebaut sein. Jedes programmierbare UND-Gatter stellt einen Produkt-Term dar. Das meistverbreitete FPLA, der Typ 82S153, verfügt über 42 Produkt-Terme. 20polige mittlere PAL-Bausteine (z. B. 16L8 oder 16R8) haben 64 Produkt-Terme. Der noch recht junge Typ 22V10 besitzt ein Dioden-Array mit 5808 Durchschmelzverbindungen für insgesamt 132 Produkt-Terme. Von diesen sind 120 für den Logikaufbau und 12 für architektonische oder Steuer-Terme vorgesehen, wobei eine noch später zu beschreibende variable Ausgangs-Makrozellenkonfiguration programmiert wird.

Jeder Produkt-Term des 22V10 besteht aus einem 44poligen UND-Gatter. Die Ausgänge dieser UND-Gatter sind mit festverdrahteten ODER-Gattern verbunden. Produkt-Terme sind den ODER-Gattern zugeordnet in einer variablen Verteilung innerhalb des Bausteins mit einer Breite von 8...16. Damit lassen sich bis zu

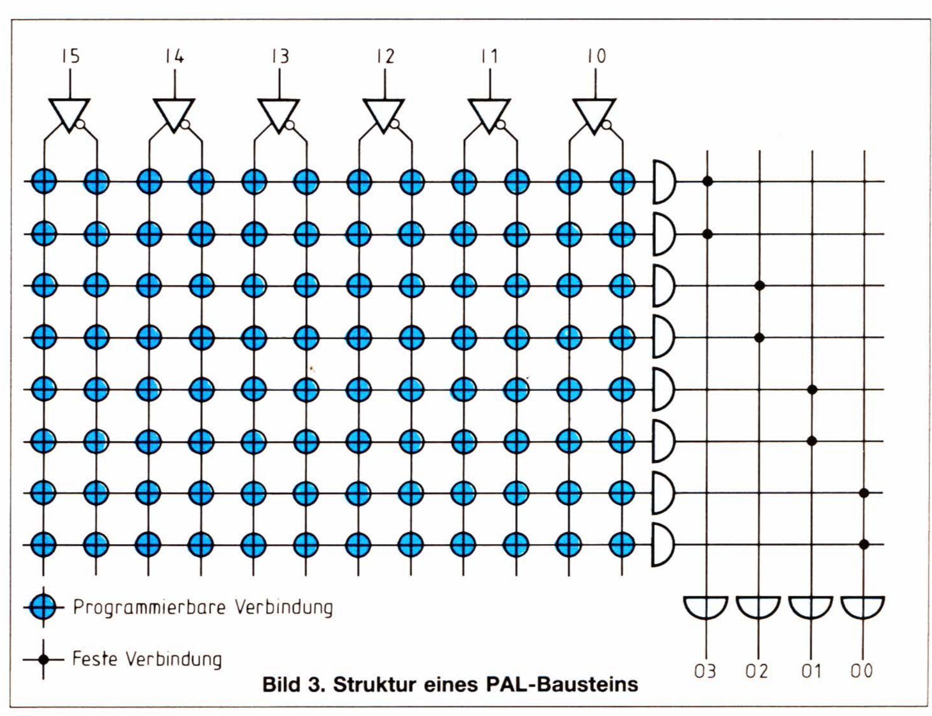


Bild 3. Struktur eines PAL-Bausteins

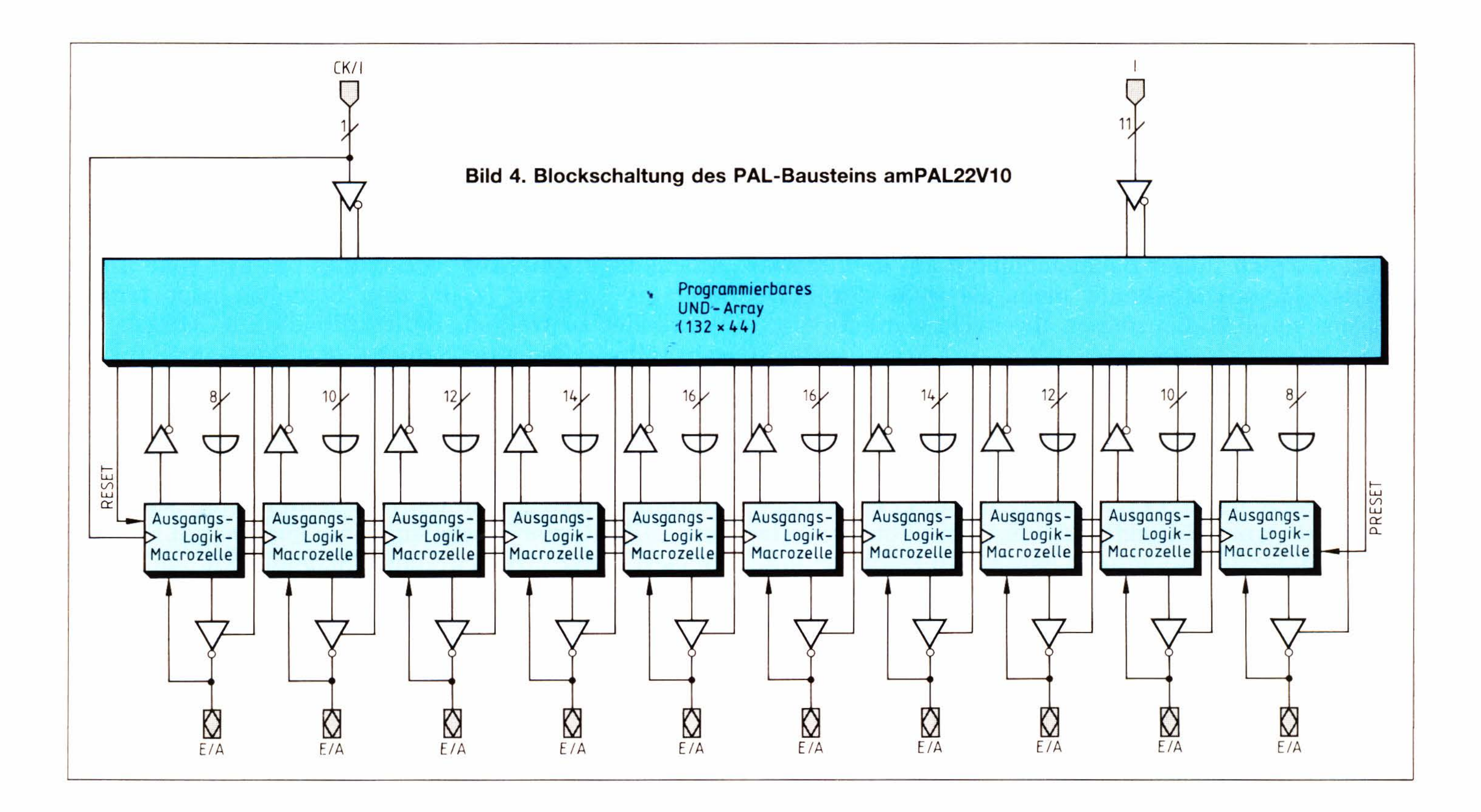


Bild 4. Blockschaltung des PAL-Bausteins amPAL22V10

16 logische Terme an einem Ausgang in einem einzigen Taktzyklus ansprechen. Die 44 Produkt-Terme der 20poligen PAL-Elemente sind den ODER-Gattern mit einer Breite von 8 zugeordnet.

In diesen PAL-Bausteinen sind die ODER-Gatter entweder kombinatorisch (die Typen 16L8, 16R8) oder über Register (16R8) mit dem Ausgang verbunden. In der Registerversion werden die Daten an die UND-Matrix intern vom Ausgang Q zurückgeführt (*Bild 5b*), was außerordentlich hilfreich bei der Auslegung von Zustands-Maschinen ist. Das 8 Bit breite Produkt-Term-ODER-Gatter gibt zusammen mit dem Register und dem Rückkopplungspfad etwa 60 Random-Gatter. Eine Extrapolierung dieses Wertes auf 8-, 10-, 12-, 14- und 16-Bit-ODER-Gatter ergibt beim 22V10 eine Zahl von 900 äquivalenten Gattern.

Allerdings ist schwierig, eine allgemeine Regel zur Angabe des Wertes der äquivalenten Gatter für ein PLD anzugeben. Diese Zahl hängt davon ab, wie gut eine Logik mit Hilfe der UND-ODER-Konfiguration aufgebaut werden kann. Weil das bei vielen Entwürfen nicht in der effizientesten Weise möglich ist, sollte der Typ 22V10 als universelles Bauelement mit 600 bis 700 äquivalenten Gatterfunktionen in Erwägung gezogen werden. Bisherige PAL-Typen, z. B. 16R8, liegen in einem Bereich von 150 bis 300 verwendbaren Gattern.

Ein besonderes Merkmal des 22V10 ist die Ausgangs-Logik-Makrozelle, die die feste kombinatorische oder sequentielle Ausgangskonfiguration ersetzt. Jede Makrozelle kann individuell programmiert werden, wobei vier verschiedene Betriebsarten zur Verfügung stehen: mit Register (aktiv High oder aktiv Low) oder kombinatorisch (aktiv High oder aktiv Low). Diese Flexibilität ist sehr hilfreich für komplexe Steuersignalerzeugung, Codierung und Decodierung.

Alle derzeit verfügbaren PLD-Bausteine werden in bipolarer TTL-Technologie gefertigt. Sperrschicht-Isolations-Prozesse werden von der Oxid-Isolation in den meisten neuen Entwürfen ersetzt. Datenblätter zeigen Verzögerungszeiten vom Ein- zum Ausgang im Bereich von maximal 25...35 ns. Typische Verzögerungen für die schnelleren Typen liegen bei 12 ns. Um diese Logik mit MSI/SSI-TTL-Elementen aufzubauen, müßte man sechs Gatterstufen hintereinander anordnen. Dabei rechnet man mit entsprechenden Verzögerungszeiten einschließlich der externen Treiber mit 2...3 ns pro Gatter. Die Leistungsaufnahme beträgt typisch 330 mW bei einem PAL-Baustein mit 64 Produkt-Termen und 600 mW für einen Typ mit 130 Produkt-Termen. Umgerechnet auf das einzelne Gatter liegt somit die Verlustleistung in einem Bereich zwischen 0,5 und 1,5 mW.

Logistik

Eine einfache kundenspezifische Logikschaltung läßt sich in weniger als 1 Std. definieren, in das System eingeben, simulieren sowie als Prototyp realisieren. Dafür sind Geräteinvestitionen erforderlich, die weniger als 25 000 DM betragen. Weil Veränderungen oder Korrekturen unverzüglich auszuführen sind und neue Bausteine sich ohne zeitliche Verzögerung herstellen lassen, fördern PLDs interaktive Entwurfsmethoden. Mit PLDs läßt sich Silizium-Hardware mit derselben Einrichtung

herstellen, auf der Programmierer Software schreiben. Die Elemente durchlaufen in kurzer Zeit verschiedene Prototypen-Versionen. Eine vollständige Systemlogik läßt sich testen und modifizieren, bis sie wie gewünscht arbeitet. Dies ist bei anderen Halb-Standard-Bauelemententypen nicht so einfach möglich, weil für diese eine Überarbeitungs-Zyklus-Dauer von 8 bis 10 Wochen erforderlich ist, wobei die Kosten für diese Arbeiten bei jeweils 20 000 bis 100 000 DM liegen können.

Ein zweiter Vorteil einer PLD-Lösung liegt darin, das Verhalten einer spezifischen Schaltung in einem Versuchsaufbau erproben zu können. Obwohl Software für die Gleich- und Wechselspannungsanalyse beim IC-Entwurf immer mehr verbessert werden, gibt es immer noch viele Eigenschaften, z. B. Verzögerungen in kritischen Pfaden, Schalt-Störpegel und Leitungs-Reflexion, die ein Computerprogramm nicht genau simulieren kann. Bei PLDs läßt sich dieses bereits an einem System-Prototyp untersuchen; korrigierende Maßnahmen erfordern keine teuren Modifizierungszyklen.

Auch völlig neue Systemarchitekturen wurden mit PLDs ausprobiert. Wenn ein solches System sich als funktionsfähig erweist, läßt es sich sehr leicht in komplexen „Semi Custom"-Bauelemente unterbringen. Bei geringen Produktionsstückzahlen – z. B. 1000 oder weniger pro Jahr – ist es in vielen Fällen wirtschaftlicher, bei der PLD-Implementierung zu bleiben, weil dann keine zusätzlichen Investitionskosten anfallen. *Bild 6* zeigt die Kurven, mit denen sich die Wirtschaftlichkeit eines Bauelementetyps bestimmen läßt.

Eine weitere Unterscheidung zwischen PLDs und anderen kundenspezifischen Bauelementen sind die Zweithersteller. Weil es sich bei PAL-Bauelementen um standardmäßig herzustellende Halbleiter handelt, gibt es

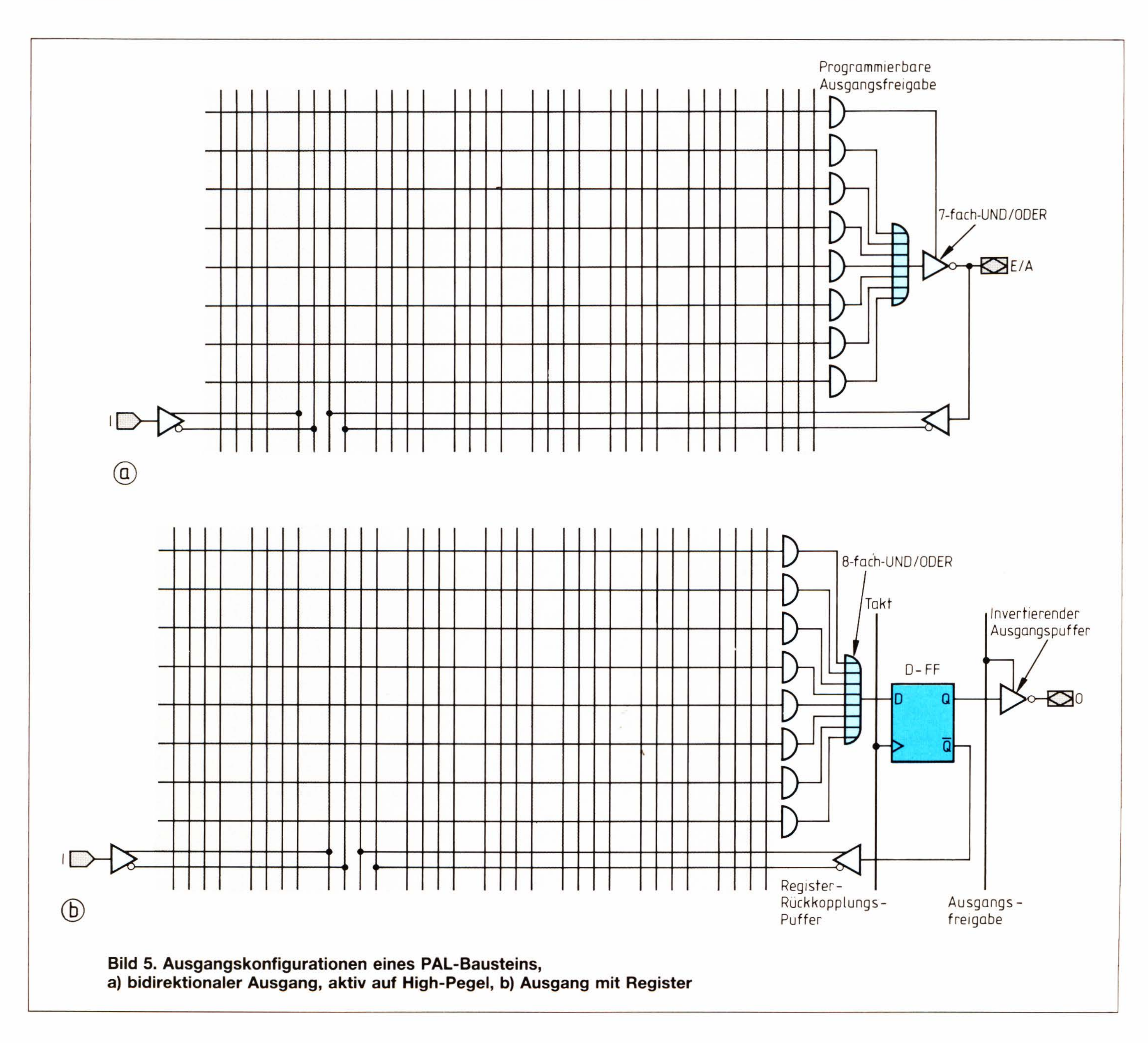


Bild 5. Ausgangskonfigurationen eines PAL-Bausteins, a) bidirektionaler Ausgang, aktiv auf High-Pegel, b) Ausgang mit Register

auch für viele Typen Zweitlieferanten. Dies ist für viele kritische Produkte ein entscheidender Punkt.

Systementwurf

Programmierbare Logikbausteine werden heute in zwei Bereichen des Systementwurfes verwendet. In kleinen Systemen verfügen PLDs sehr häufig über ausreichend Logikfunktionen, um die wichtigsten Funktionsblöcke einer Maschine zu realisieren. Manchmal werden sie alleine verwendet, häufiger allerdings in Zusammenhang mit Standard- oder anderen kundenspezifischen Bauelementen. In größeren Systemen stellen PLDs die verbindenden Funktionen her, durch die komplexe Mikroprozessor-Peripheriebausteine oder Gate-Array-LSI-Funktionen miteinander verknüpft sind.

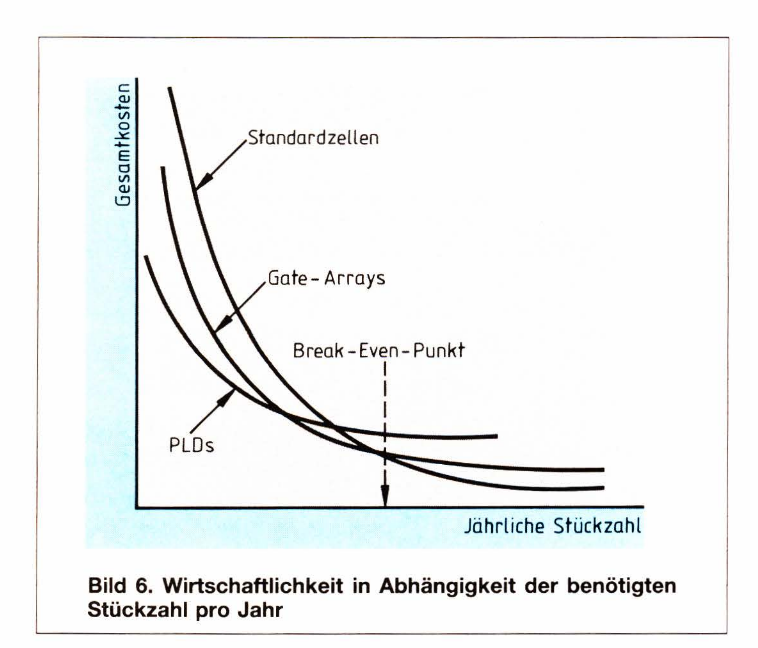


Bild 6. Wirtschaftlichkeit in Abhängigkeit der benötigten Stückzahl pro Jahr

Jedes digitale System läßt sich in drei Hauptfunktionsblöcke einteilen: Steuerpfad, Datenpfad und Interfaces. PLDs lassen sich in allen drei Bereichen anwenden.

Die Steuereinheit interpretiert Makro-Instruktionen, die sich im Hauptspeicher befinden, und führt diese aus. Außerdem erzeugt sie die Signale zur zeitlichen und sequentiellen Ansteuerung sowie die Signale, mit denen Entscheidungen ausgelöst werden. PLDs sorgen in diesem Funktionsbereich dafür, die Anzahl der normalerweise erforderlichen SSI/MSI-Logikbausteine zu verringern. In mikroprogrammierten Implementierungen lassen sich PLDs direkt als Zustands-Maschinen mit bis zu 48 Zuständen programmieren. Auch können sie als Bestandteile eines Adreß-Sequenzers, der von einem PROM gesteuert wird, dienen.

PLDs eignen sich zum Aufbau von Befehlsregistern, Programmzählern, Befehlsdecodierern, Befehlssequenzern sowie zum Erzeugen von Steuer-/Zeitsignalen zur Ausführung von Instruktionen. Das flexible ODER-Array eines FPLA-Bausteins eignet sich sehr gut für einfache oder auch komplexe Befehlsdecodierung. PAL-Bausteine lassen sich ebenfalls zum Decodieren von Befehlen verwenden.

Datenpfad-Funktionen sind z. B. die Manipulation (ALUs), das Speichern (Register-Files und Pipeline-Register), die Auswahl (Multiplex), Steuerung und das Schieben (z. B. Barrel-Shifter). Beim Datenpfad handelt es sich typischerweise um den am meisten strukturierten Teil des Systems, der sehr früh im Entwurfszyklus definiert werden muß. Im Gegensatz zur Steuerung läßt sich der Datenpfad nicht so einfach verändern. Leistung und Schaltungsdichte sind die kritischsten Punkte dieses Funktionsblocks. Aus diesem Grund stellen Standard-VLSI-Bausteine oder Gate-Arrays hoher Schaltungsdichte und hoher Leistung die beste Lösung dar.

Allerdings können, wenn die Stückzahl gering ist und eine kundenspezifische Architektur gewünscht wird, PLDs eine brauchbare Lösung sein, insbesondere bei genau definierten Datenpfad-Funktionen, z. B. beim Barrel-Shifter (siehe *Kasten*). Bei diesem Beispiel zeigt sich, daß die geforderte Funktion sich ideal für die Architektur von PAL-Bauelementen eignet.

Seit ihrer Markteinführung hat man PLDs in Interface-Schaltungen verwendet, z. B. als Zustands-Sequenzer oder zur Steuerung externer Logik für die Adreß-Multiplex-Umschaltung, Zeitsignalerzeugung und Refresh-Steuerung. In diesen Anwendungen ersetzen PLDs etwa 15 bis 20 SSI/MSI-Bausteine. Bei anderen Interface-Anwendungen, z. B. dem Anschluß von Mikroprozessoren an Peripherieeinheiten, vereinfacht sich aufgrund der Flexibilität von PLDs jede Veränderung oder Erweiterung, weil lediglich eine Neuprogrammierung des Bausteins erforderlich ist. PLDs lassen sich auch für den Aufbau einer Bus-Arbitrations- und Request-Synchronisations-Logik verwenden.

Der Ersatz einer existierenden, bewährten SSI/MSI-Logik oder die Implementierung einer völlig neuen Schaltungskonfiguration mit PLDs erfordert zwei verschiedene Wege der Realisierung. Beim Ersetzen einer existierenden Schaltung kommt es darauf an, möglichst viele Bausteine durch den PLD-Chip überflüssig zu machen. Ein solches Modul kann entweder nur kombinatorisch, nur sequentiell oder eine Kombination aus kombinatorischer und sequentieller Logik sein. Ein Entwickler geht folgendermaßen vor:

- Kombination zusammenhängender MSI/SSI-Funktionen von etwa 5 bis 10 Bausteinen in einem Modul
- Festlegung der erforderlichen Eingangs-/Ausgangs-Anschlußzahl des gesamten Funktionsblocks
- Entwicklung der logischen Gleichungen für dieses Modul
- Reduzierung von Eingangs-/Ausgangs- und Produkt-Termen mit Hilfe der Karnaugh-Tabelle oder anderer Logik-Minimierungsverfahren
- Wenn alle Kerne in minimaler Form vorliegen, überprüft man, ob sie von verfügbaren PLD-Typen erfüllt werden können.
- Wenn ein einzelner PLD-Baustein nicht ausreicht, muß man die Funktion erneut aufteilen oder die Pro-

dukt-Term-Erweiterungsmöglichkeiten bestimmter PLDs nutzen.

– Wenn die Anforderungen erfüllt werden und der PLD-Typ nicht effizient ausgenutzt ist (z. B. in bezug auf E/A-Anschlüsse, Produkt-Terme usw.), versucht man, noch mehr SSI/MSI-Chips im Modul unterzubringen.

Reicht die Anzahl von Produkt-Termen, die von einem einzelnen FPLA- oder PAL-Baustein geboten wird, nicht aus, läßt sich diese auch durch eine weitere Schaltung mit identischen Eingängen erweitern, wobei die entsprechenden Ausgänge miteinander verbunden werden (*Bild 7a*). Überschreitet die Anzahl der Ein-

Entwurf einer Logikschaltung mit PAL-Bausteinen

Die UND/ODER-Struktur von PAL-Bausteinen bedeutet, daß mit ihnen jede Logikfunktion implementierbar ist, die sich in Form der Summe von Produkten darstellen läßt, so lange die erforderliche Zahl von Eingängen, Ausgängen und Produkt-Termen nicht die Kapazität des betreffenden Bausteines überschreitet. Am Beispiel des Typs AmPAL22V10 soll die Implementierung eines 8-Bit-Barrel-Shifters mit Registern gezeigt werden.

Anforderungen an das Endprodukt

Ein Barrel-Shifter läßt eingegebene Daten zyklisch über eine bestimmte Zahl von Bitpositionen rotieren. Dieser zyklische Rotiervorgang erfolgt so, daß die Daten vom höchstwertigen Ende des Schieberegisters zum niedrigwertigen Ende zurückgeführt werden.

Ein 8-Bit-Barrel-Shifter mit Register erfordert wenigstens acht Dateneingänge, acht Ausgänge mit Registern, drei Steuerleitungen, mit denen die Anzahl der Schiebepositionen festgelegt wird, einen Takteingang, einen Chipfreigabeeingang sowie einen Ausgabe-Freigabe-Anschluß zur Steuerung des Tri-State-Puffers am Register.

Ausgeführte Schaltungen

Der erste Schritt beim PAL-Entwurf ist die Aufstellung der Boole'schen Gleichungen für diese Funktion. Eine Übersicht gibt *Tabelle 1*. Folgende Symbole werden für die logischen Gleichungen verwendet:

* = UND
+ = ODER
/ = NICHT (Inversion).

Boole'sche Gleichungen für zwei der acht Ausgänge (/Y7 und /Y6) des 8-Bit-Barrel-Shifters mit Register sehen folgendermaßen aus:

```
/Y6 := /CE * /S2 * /S1 * /S0 * /D6 +
       /CE * /S2 * /S1 * S0 * /D5 +
       /CE * /S2 * S1 * /S0 * /D4 +
       /CE * /S2 * S1 * S0 * /D3 +
       /CE * S2 * /S1 * /S0 * /D2 +
       /CE * S2 * /S1 * S0 * /D1 +
       /CE * S2 * S1 * /S0 * /D0 +
       /CE * S2 * S1 * S0 * /D7
/Y7 := /CE * /S2 * /S1 * /S0 * /D7 +
       /CE * /S2 * /S1 * S0 * /D6 +
       /CE * /S2 * S1 * /S0 * /D5 +
       /CE * /S2 * S1 * S0 * /D4 +
       /CE * S2 * /S1 * /S0 * /D3 +
       /CE * S2 * /S1 * S0 * /D2 +
       /CE * S2 * S1 * /S0 * /D1 +
       /CE * S2 * S1 * S0 * /D0
```

Die sechs restlichen Ausdrücke lassen sich ähnlich erzeugen. Man erkennt, daß ein Baustein mit wenigstens 76 Produkt-Termen erforderlich ist, 64 Produkt-Terme werden für die Logikfunktionen und zwölf zusätzliche Logik-Terme zur Realisierung der Ausgangs-Freigabe (acht für Ausgabe, vier für Eingabe) benötigt.

Der Barrel-Shifter mit Register verschiebt acht Datenbits synchron mit dem Taktsignal in die Ausgangsregister, die von

Tabelle 1. Funktionsdefinitionen

Steuereingänge			Ein-/Ausgangstabelle							
S2	S1	S0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	D6	D5	D4	D3	D2	D1	D0	D7
0	1	0	D5	D4	D3	D2	D1	D0	D7	D6
0	1	1	D4	D3	D2	D1	D0	D7	D6	D5
1	0	0	D3	D2	D1	D0	D7	D6	D5	D4
1	0	1	D2	D1	D0	D7	D6	D5	D4	D3
1	1	0	D1	D0	D7	D6	D5	D4	D3	D2
1	1	1	D0	D7	D6	D5	D4	D3	D2	D1

den drei Steuerbits (S2 bis S0) spezifiziert sind. Dazu muß der Chip-Enable-Anschluß auf Low liegen. Wenn dieser auf High-Potential liegt, stehen im Ausgangsregister nur Einsen. Bei Low-Pegel auf der Ausgangs-Freigabeleitung ist der Zustand der Ausgänge hochohmig.

Der nächste Schritt ist die Auswahl eines bestimmten PAL-Bausteins für die Funktion. Wie bereits vorher spezifiziert, erfordert die Barrel-Shifter-Funktion insgesamt 14 Eingänge und acht Ausgänge.

Der Baustein amPAL22V10 verfügt über 22 Eingänge und 10 Ausgänge – damit stehen genügend Anschlüsse zur Verfügung. Die Einrichtung zum dynamischen Programmieren der E/A-Funktionen des 22V10 ermöglichen die Anpassung an die gestellten Anforderungen. Auch die 76 Produkt-Terme für diese Funktion lassen sich vom 22V10 realisieren. Dieser Baustein kann maximal 132 Produkt-Terme zur Verfügung stellen. In bezug auf die Ein- und Ausgänge sowie die Produkt-Terme wird der Chip zu 13 % ausgenutzt.

Um den 22V10 zu 100 % nutzen zu können, muß eine Logikfunktion realisiert werden, die zwei Ausgänge mit acht Produkt-Termen, zwei Ausgänge mit zehn Produkt-Termen, zwei Ausgänge mit zwölf Produkt-Termen, zwei Ausgänge mit 14 Produkt-Termen und zwei Ausgänge mit 16 Produkt-Termen bestehen. Jedes Produkt-Term ist eine Funktion von maximal 22 Eingangssignalen. Dies zeigt, wie umfangreich und komplex Boole'sche Gleichungen sein können, die sich mit dem 22V10 implementieren lassen.

gangsvariablen die Anzahl der Eingänge, läßt sich die Gesamtzahl der Eingänge auf eine Schaltung mit mehreren PLDs aufteilen. Dies erreicht man, indem man die Eingänge in separate Gruppen aufteilt (*Bild 7b*). Eine Erweiterung der Zahl von Ausgängen läßt sich sehr einfach durch Verbinden von Eingangsanschlüssen erreichen.

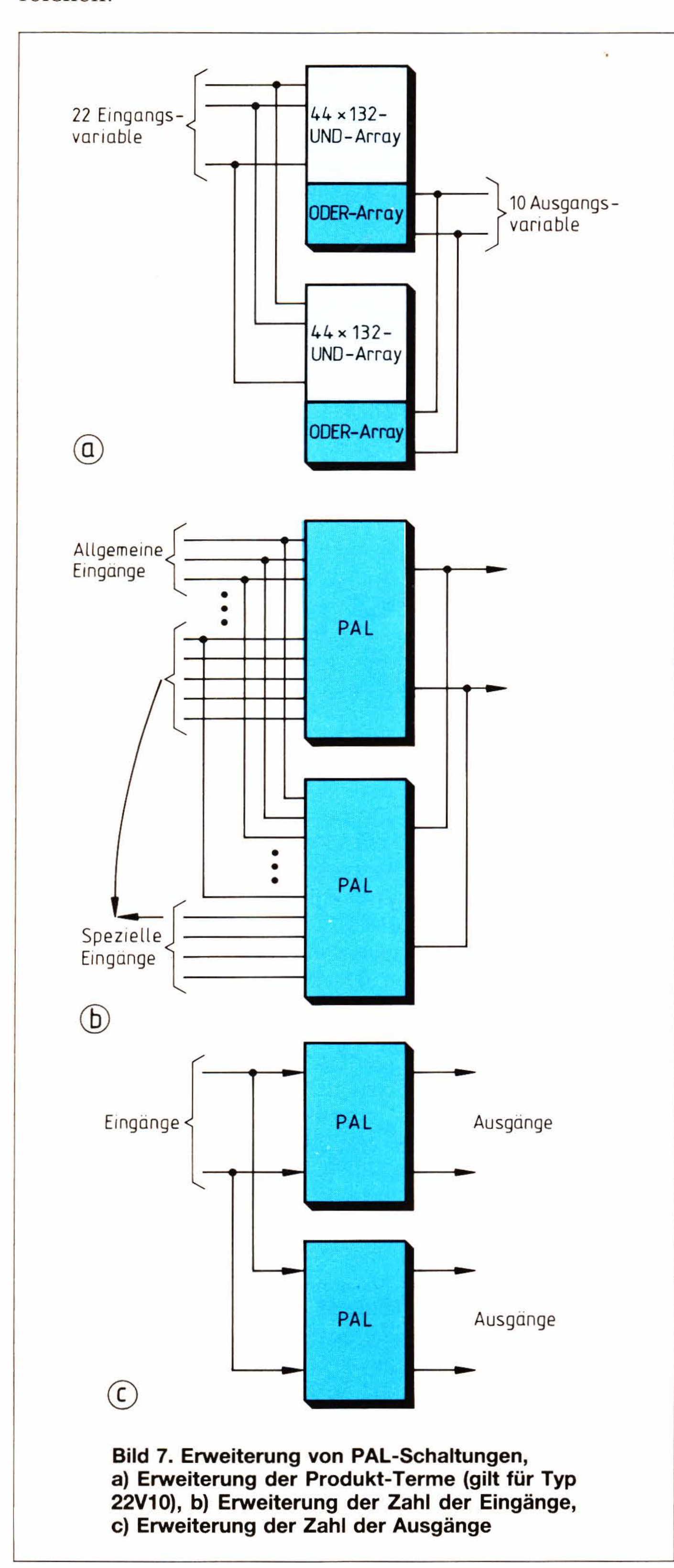


Bild 7. Erweiterung von PAL-Schaltungen, a) Erweiterung der Produkt-Terme (gilt für Typ 22V10), b) Erweiterung der Zahl der Eingänge, c) Erweiterung der Zahl der Ausgänge

Zur vollständigen Ausnutzung der PLD-Funktionsmerkmale sollte bei einer Neuentwicklung auf einen hierarchischen Top-Down-Entwurf hingearbeitet werden. Der erste Schritt ist die Aufteilung der Logikfunktionen in einen Steuer-, Daten- oder Interface-Pfad. Danach sind dafür die logischen Gleichungen aufzustellen und Eingangs-/Ausgangs- sowie Produkt-Term-Minimierung vorzunehmen. Anschließend ist zu versuchen, die so entstandene Schaltung mit Hilfe eines PLD-Bausteins zu verwirklichen.

Um die gegebene Schaltung an eine PLD anzupassen, sollte der Entwickler den PLD-Chip nach folgenden architektonischen Merkmalen aussuchen:

- E/A-Struktur. Dazu zählt die Gesamtzahl der E/A-Anschlüsse und deren Auslegung, z. B. ob sie nur Ein- und Ausgänge sind oder sich bidirektional programmieren lassen.
- Konfigurierbarkeit/Programmierbarkeit der E/A-Anschlüsse. Dazu zählt die Programmierbarkeit der Ausgangs-Anschlüsse (entweder statisch oder dynamisch) als Ausgänge mit Register oder Latches oder mit kombinatorischer Logik oder die Programmierung der Polarität, d. h. aktiv auf High- oder Low-Pegel.
- Flexibilität der Produkt-Terme. Dies bedeutet die Gesamtzahl der Produkt-Terme, die kleinste Zahl der Produkt-Terme pro Ausgang und deren Verteilung.
- Flexibilität der Register-Konfiguration. Dies bedeutet die Anzahl der Eingänge/Ausgänge mit Register, ob die Register ein gemeinsames oder separate Taktsignale haben und ob die Ausgänge der Register sich einzeln oder gemeinsam freigeben lassen.
- Treiberkapazität. Dieser Punkt ist insbesondere für Interface-Anwendungen von Bedeutung.

Software-Hilfsmittel für den Entwurf

Läßt sich die gewünschte Logikfunktion in einer PLD-Schaltung realisieren, müssen die Gleichungen so in ein Muster umgesetzt werden, daß ein Programmiergerät weiß, welche Verbindung durchgeschmolzen werden muß.

Das erste Computerprogramm, das diese Aufgabe übernehmen konnte, trägt den Namen „PALASM" (PAL-Assembler). Unterschiedliche Versionen dieser Software existieren für verschiedene Computer, die im Ingenieurbereich Anwendung finden, z. B. IBM-Großrechner, VAX und PDP-11 von DEC sowie viele Personal Computer, die unter CP/M oder MS-DOS arbeiten.

PALASM akzeptiert als Eingabe Boole'sche Gleichungen und assembliert diese Daten in eine „Fuse-Map", die direkt für die Ansteuerung von PLD-Programmiergeräten geeignet ist. Außerdem lassen sich Testvektoren eingeben, die für Simulation und Debugging der eingegebenen Gleichungen sowie für den Test des fertiggestellten Bausteins geeignet sind. Die eingegebenen Daten stellen zusammen mit den Software-Dokumentations-Ausgaben eine standardisierte Form der Dokumentation dar.

PALASM läßt sich auch mit Hilfe von EPROMs direkt in einer Anzahl von PLD-Programmiergeräten unterbringen. Mit Hilfe eines „dummen" Terminals kann man dann die Gleichungen direkt in das Programmiergerät eingeben.

Eine Anzahl von PLD-Entwurfs-Unterstützungsprogrammen in höheren Programmiersprachen, die auf Compilern basieren, sind bereits verfügbar bzw. werden derzeit entwickelt. Das erste kommerziell erhältliche Produkt ist „CUPL", ein Programmpaket, das auf UNIX läuft und in C geschrieben ist (Assisted Technology). Von Data I/O kommt ein Produkt mit der Bezeichnung „ABEL" und Advanced Micro Devices wird bald das Paket „PLPL" vorstellen. Diese neue Generation der Software vermeidet viele Fehler, die in früheren Programmpaketen auftraten. Außerdem verbessert sie die Entwurfsproduktivität, weil sie die Verwendung von Makros, freien Kommentaren und Klammerfunktionen erlauben.

Zukünftige Entwicklungen

Neue PLD-Produkte, die sich derzeit in Entwicklung befinden, erfüllen verschiedene Anforderungen des Systementwurfs. Dazu gehören z. B. erhöhte Arbeitsgeschwindigkeit, geringere Stromaufnahme, erweiterte E/A-Funktionen, höhere funktionelle Flexibilität sowie größere Logikkomplexität.

Verbesserte Leistungen erreicht man, wenn man moderne fotolithografische Verfahren beim herkömmlichen Bipolarprozeß anwendet. Damit lassen sich PAL-Bausteine mit Verzögerungszeiten von 15 ns herstellen, in den nächsten Jahren vielleicht sogar mit Verzögerungszeiten von 10 ns oder weniger.

Verschiedene Hersteller entwickeln ECL-Typen, für die eine Verzögerungszeit von 5 ns spezifiziert ist. Diese Typen werden eine wertvolle Ergänzung für ECL-Gate-Arrays darstellen, die in zunehmendem Maße in großen Systemen Verwendung finden.

Die gleichen Verbesserungen der Technologie, die die Geschwindigkeit erhöhen, sorgen auch dafür, daß herkömmliche Versionen der PAL-Bausteine weniger Leistung aufnehmen. Man rechnet in den nächsten zwei Jahren mit einer Verringerung der Leistungsaufnahme um 50 %.

CMOS-Technologie führt einerseits dazu, daß Bausteine weniger Leistung aufnehmen, andererseits, daß sich wesentlich mehr Gatterfunktionen, z. B. mehr als 1000, auf einem Chip unterbringen lassen. Allerdings stellen programmierbare CMOS-Bausteine die Entwickler vor neue Probleme, weil sie z. B. 30 oder mehr Eingänge pro Gatter besitzen. Derzeit arbeitet man auch an nichtflüchtigen Logik-Elementen in CMOS-Technik, die sich sowohl nur einmal als auch innerhalb eines Systems neu programmieren lassen.

Neue Trends betreffen auch die Architektur programmierbarer Bausteine. Ein typisches Beispiel ist der Typ 22V10. Dessen Ausgänge lassen sich statisch oder dynamisch, mit Register, Latches oder mit kombinatorischen Funktionen, aktiv auf High- oder Low-Pegel konfigurieren. Die Verteilung der Produkt-Terme läßt sich dynamisch zwischen den Ausgängen verteilen. Zusätzliche Merkmale, z. B. programmierbare Produkt-Freigabe-Terme für Reset/Preset für die Ausgangsregister und Ausgangs-Freigabeleitungen sind in Kürze zu erwarten. Diagnoseschaltungen, z. B. das serielle Schattenregister von AMD (SSR) unterstützen Baustein- und System-Test.

Derzeit erhältliche Bausteine besitzen eine relativ geringe Komplexität, weil die meisten Anbieter sich auf kleine Chipgrößen mit begrenzten E/A-Funktionen beschränken, um die Chips in den weitverbreiteten schmalen 20- und 24poligen Gehäusen unterbringen zu können. Diese waren ursprünglich für den Ersatz von TTL-SSI/MSI-Funktionen gedacht. Verbesserte E/A-Funktionen und höhere Komplexität bipolarer PLDs führen zu 40poligen DIL-Gehäusen und 84poligen Chip-Carriern. Diese bieten bis zu 32 Eingänge/16 Ausgänge bzw. 64 Eingänge/32 Ausgänge. Der derzeitige technische Stand logischer Komplexität von 120 Produkt-Termen wird bis zum Ende dieser Dekade um den Faktor 10 bis 20 erhöht sein. Obwohl diese Größenordnung die Schaltungsdichte von heutigen Gate-Arrays überschreitet, werden PLDs auch in Zukunft sich in gleicher Weise von maskenprogrammierbaren Bauelementen unterscheiden, weil bei PLDs die Programmier- und Test-Schaltungen mit auf dem Chip integriert sein müssen. Dieser Schaltungs-Überbau nimmt 10 bis 20 % der aktiven Chipfläche ein.

(Übersetzung: Be)

Literatur

[1] *Kitson, J., Ow-Wing, K.*: The Berkeley-1 Plus – A high performance CPU. Programmable Array Logic Handbook, Advanced Micro Devices, S. 5...72.

[2] *Kitson, B., Rosen, J. B.*: Logical Alternatives in Supermini Design. Computer Design, Nov. 1983.

[3] *Louie, G., Ho, T., Cheng, E.*: The MicroVAX 1 Data-Path Chip. VLSI Design, Dez. 1983.

[4] *Beresford, R.*: Comparing Gate Arrays and Standard-Cell ICs. VLSI Design, Dez. 1983.

Willibald Voldan

Technische und wirtschaftliche Aspekte

Schnelle und komplexe Recheneinheiten mit immer größeren Speichereinheiten benötigen leistungsstarke Peripheriebauelemente. Noch dazu geht der Trend eindeutig von einem großen Gesamtsystem hin zu modular aufgebauten Subsystemen. Das ist der Zeitpunkt für schnelle, flexible Lösungen mit programmierbarer Logik. Innerhalb dieser Familien stellt die PAL-(Programmable Array Logic)Familie aus technischen und ökonomischen Gründen sicherlich die treibende Kraft hinter diesem Logikkonzept dar.

Bei der Realisierung digitaler Systeme war der Anwender lange Zeit auf konventionelle TTL-Bauelemente (SSI/MSI) angewiesen, die jeweils nur einige Funktionen (zum Beispiel 4-NAND-Gatter 7400 oder 4-Bit-Counter 74161) enthalten. Für komplexe logische Anwendungen bietet sich zwar der Mikroprozessor an, doch kommt auch er nicht ohne zusätzliche Logik und Software aus. Er hat darüber hinaus den Nachteil, wesentlich langsamer zu sein. Auch benötigen LSI- und VLSI-Bauelemente immer wieder bestimmte logische Funktionen, um sie zu Systemen zusammenzuschalten.

Der Entwicklungsingenieur kann heute aus einem großen Spektrum von Bauelementen wählen, um seine digitalen Schaltungen zu realisieren. Dabei sind grundsätzlich zwei Gruppen zu unterscheiden (*Bild 1*):

- *Standardprodukte:* Die Architektur wird ausschließlich vom Hersteller festgelegt und kann vom Anwender nicht mehr verändert werden. Beispiele dafür sind Mikroprozessoren mit festem Instruktionssatz und deren Peripherie, mikroprogrammierbare Bauelemente sowie bipolare und MOS-SSI/MSI-Bausteine.
- *Kundenschaltungen:* Der Hersteller produziert eine vom Anwender vorgegebene Funktion. Abhängig davon, inwieweit der Schaltkreis bereits vorstrukturiert wurde, unterscheidet man Vollkunden- und Semikundenschaltungen, die dann wieder in Gate-Arrays und programmierbare Logik unterteilt werden können.

Jedes dieser Konzepte hat seine spezifischen Eigenschaften, Vor- und Nachteile hinsichtlich Kosten, Verfügbarkeit, Architektur und Flexibilität.

In vielen Schaltungen können mehrere dieser Konzepte gemeinsam vorkommen. So werden zum Beispiel PALs oft als Peripherie um Gate-Arrays herum verwendet, um diese Schaltkreise flexibel zu halten.

Standardprodukte

Die ersten integrierten Schaltkreise der bekannten Logikfamilien reduzierten die bisher mit Transistoren, Dioden und Widerständen diskret aufgebauten logischen Schaltungen auf Standard-Gatterfunktionen mit gleichzeitiger Normung des Logikkonzepts. AND, OR, NAND, NOR, INVERTER, EXCLUSIVE OR und Flipflops in Standard-TTL-Technik entstanden mit einheitlichen elektrischen Spezifikationen (Fan-in, Fan-out, Störabstand usw.). Bis zu diesem Zeitpunkt standen folgende Logikkonzepte zur Diskussion:
RTL (Widerstand-Transistor-Logik), CTL (Complementär-Transistor-Logik), TTL (Transistor-Transistor-Logik).

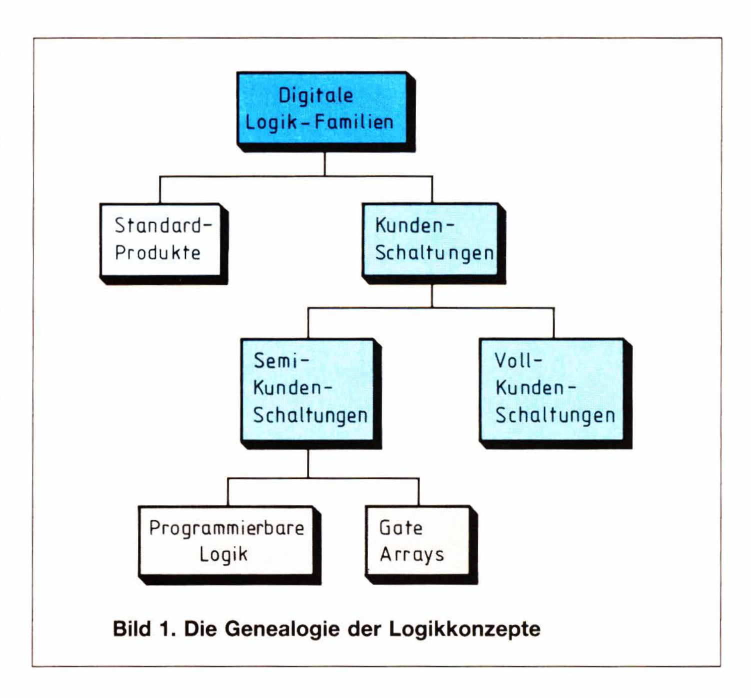


Bild 1. Die Genealogie der Logikkonzepte

Die Industrie hat sich für die TTL-Technik entschieden und damit richtungsweisend die Weiterentwicklung dieses Konzepts beeinflußt.

Die nächste Integrationsstufe von SSI-(Small Scale Integration)Bausteinen war zwangsläufig mit der Notwendigkeit gegeben, Funktionen wie Addierer, Schieberegister, Zähler und Multiplexer aufzubauen; dies führte zu MSI-(Medium Scale Integration)Bausteinen.

Verfeinerte Strukturen und größere Silizium-Waferdurchmesser von 3 und 4 Zoll eröffneten schließlich die Möglichkeit, LSI-(Large Scale Integration)Bausteine wirtschaftlich zu fertigen.

Die fertigungsbedingte Notwendigkeit, Standardschaltungen zu liefern, ist mit LSI-Schaltungen, die definierte logische Standardfunktionen zusammenfassen, ermöglicht worden. Diese hochintegrierten Bausteine realisieren eine Vielzahl der früher mühsam mit den SSI/MSI-Familien aufgebauten Schaltungen, können jedoch den spezifischen Anforderungen bestimmter Applikationen nur wieder über zusätzliche, niedrig integrierte Logik angepaßt werden.

- Standardprodukte benötigen verhältnismäßig wenig Training und Erfahrung für den Entwicklungsingenieur, um sie anwenden zu können. Sie sind sicherlich die „billigste“ Lösung, wenn man nur den Preis für das einzelne Bauelement betrachtet.
- Mehrere Hersteller für das gleiche Produkt bringen dem Anwender preisgünstige Schaltungen.
- Die mehrjährige Erfahrung mit Standardbauelementen gewährleistet dem Anwender umfangreiche Applikationsstützung.
- Die Verantwortung für die Architektur, das Testen und die Spezifikationen liegt ausschließlich beim Hersteller dieser Bauelemente. Der Prozeß zur Herstellung von Standardschaltungen ist sehr effizient, weil große Stückzahlen gleicher Produkte hergestellt werden und ihre Architektur auf optimale Ausnützung von Silizium ausgelegt ist. Die Vielzahl von Applikationsunterstützung, wie Applikationssoftware (Assembler, Simulator), Hardware (Emulator) und Literatur (Handbücher, Applikationsschriften, Bedienungshandbücher usw.) macht es einfach, mit diesen Bauelementen zu entwickeln.

Bei Standardbauelementen wird zwar die Siliziumfläche optimal genutzt, dabei werden aber die Variationen der jeweiligen Funktion nicht ausgeschöpft. Es fehlt die Eigenschaft, den Anforderungen spezifischer Applikationen nachkommen zu können. Ein Standardprodukt ist für sich allein nicht für jede Anwendung geeignet. Wegen der allgemeinen Struktur der Bausteine ist es meist nicht möglich, eine Schaltungslösung mit der optimalen Anzahl von Bauelementen zu finden. Zusätzliche Funktionen werden benötigt, um die Schaltung auf die jeweilige Anforderung anzupassen. Wird nun die Kostenrechnung über das gesamte System durchgeführt, stellt sich eine Schaltung mit Standardprodukten oft als die teuerste Problemlösung von allen Konzepten dar.

Ein weiterer wesentlicher Nachteil liegt darin, daß keine wirklich unterscheidenden Merkmale der jeweiligen Funktionen vorhanden sind, die vom Anwender zur Hervorhebung genutzt werden könnten. Jeder verwendet mit Standardprodukten letztendlich dieselben Funktionen, und damit sieht grundsätzlich ein System aus wie das andere.

Kundenschaltungen, Semi-Kundenschaltungen, Gate-Arrays

In den letzten Jahren zeigt sich eine ganz deutliche Tendenzwende bei den Halbleiterherstellern. Limitierten früher fehlende Technologien und Fertigungseinrichtungen die Verwirklichung neuer Bausteinideen, so ist es heute oft umgekehrt. Mit größer werdender Funktionsdichte und der daraus resultierenden höheren Anzahl von Anschlüssen verschwindet mehr und mehr der Vorteil einer breitgefächerten Verwendungsfähigkeit. Das hat zur Folge, daß es schwerer wird, Standardfunktionen zu definieren, die es über eine Vielzahl gleichartiger Anwendungen dem Hersteller ermöglichen, diesen Baustein kommerziell interessant zu halten.

Wenige Anwendungen für eine Standardschaltung bedeuten niedrigere Stückzahlen, die wieder weniger Anreiz für Second-Source-Partner (Zweitlieferanten) bieten. Damit wird dieses Produkt einen höheren Preis, mangelhafte Applikationsunterstützung, wenig Literatur usw. aufweisen.

Man kann heute absehen, daß es im Laufe der nächsten Dekade technologisch möglich sein wird, die geometrischen Abmessungen eines digitalen Bauelementes auf 1/100 der heutigen Werte zu verkleinern. Geht man von den derzeitigen 3-μ-Strukturen aus, so werden Transistoren dann immer noch als logische Schaltelemente verwendbar sein, bevor man bei weiterer Verkleinerung an physikalische Grenzen wie Wärmeableitung, Tunneleffekte oder Schwankungen der Dotierung stößt, die dann solche Bauelemente funktionsuntüchtig machen. Trotzdem bedeutet eine Verkleinerung der Abmessung logischer Bauelemente auf 1/100 ihrer heutigen Größe, daß Funktionen in monolithischer Chipform mit Millionen von Transistoren möglich sein werden.

Wie aber oben ausgeführt, wird es immer schwieriger, solche höchstintegrierten Bauelemente in ihrer Funktionsweise zu definieren. Diese Problematik versuchen die Halbleiterhersteller mit Semi-Kundenschaltungen zu begegnen, um jedem Anwender die Möglichkeit zu geben, jeden auf seine spezielle Applikation zugeschnittenen Baustein zu konzipieren.

Semi-Kundenschaltungen bieten dem Systementwickler gewichtige Vorteile gegenüber Standardprodukten. Vergleicht man diese Familie mit SSI/MSI-Elementen, liegt der wesentliche Vorteil zunächst eindeutig in der Integration multipler Funktionen herkömmlicher Logik, die aber noch dazu für die speziellen Bedürfnisse der jeweiligen Anwendungen ausgelegt ist. Standard-LSI-Produkte bieten zwar denselben Vorteil, zwingen aber den Anwender dazu, mit der festgelegten Struktur

auszukommen. Kundenbausteine dagegen erlauben dem Anwender, seine Schaltung völlig an seine Problemstellung anzupassen. Diese Eigenschaft ist sicher das wesentliche Merkmal dieser Bauelemente, weil sie es möglich machen, das Endprodukt mit ganz speziellen, zum Mitbewerb unterschiedlichen Merkmalen zu versehen.

Nun wird gerade die mittelständische Industrie durch die in jüngster Zeit verstärkte Diskussion beziehungsweise die Undurchsichtigkeit technischer und wirtschaftlicher Gesichtspunkte verunsichert. Und nur ein geringer Prozentsatz dieser Anwender hat CAD-Systeme im Haus, wobei auch noch ein Großteil dieser Anlagen für die Erstellung von Gate-Arrays ungeeignet ist.

Limitierende Faktoren für den Einsatz von Gate-Arrays sind u. a. hohe Initialkosten bei der Entwicklung, die komplizierte und oftmals ungenügende Software, die Notwendigkeit relativ hoher Stückzahlen von nur einem Pattern, die geringe logische Dichte (verglichen mit LSI-Standardbauelementen) und die relativ hohen Kosten für das Bauelement selbst. Der Ingenieureinsatz für die Entwicklung eines Arrays kann außerdem die Kosten des Gesamtsystems maßgeblich in die Höhe treiben.

Der Entwicklungsingenieur ist nicht nur verantwortlich für seine Schaltung, sondern muß auch noch einen Schritt weiter gehen und sich die einzelnen Bausteine (Gate-Arrays) selbst erarbeiten und zur Produktionsreife bringen. Er trägt dabei nicht nur die Verantwortung für die Funktion, sondern auch für Layout, Optimierung und Erstellung des Testprogramms. Während mit herkömmlichen Logikbausteinen der SSI/MSI-Familien wegen ihrer festgelegten Strukturen, des unterschiedlichen Temperaturverhaltens und der verschiedenen Durchlaufzeiten durchaus funktionsfähige Schaltungen entstehen, ist es unmöglich, solche Konfigurationen auf ein Stück Silizium mit Tausenden von Gatterfunktionen zu bringen, ohne logische Probleme zu bekommen. Bei Array-Schaltungen besonders auf der Transistor-, aber auch auf der Logikebene muß der Systementwickler dann „Design Rules“, die bisher ausschließlich für den Bauelementehersteller maßgeblich waren, kennen und befolgen. Schattenzonen bezüglich Funktion und – vor allem – Zeitverhalten, die sich bei der Endjustage ausgleichen lassen, sind hier nicht mehr erlaubt.

Nur das Konzept strukturierter Logik verspricht bei diesen hohen Integrationsstufen einen relativ schnellen Erfolg, also ohne Redesign, insbesondere dann, wenn nicht nur geradlinige kombinatorische Logik, sondern auch sequentielle Logik integriert werden soll. Treten während der Entwicklung eines Gate-Arrays Fehler auf, so kann es mehrere Monate oder oft bis zu einem Jahr dauern, bis diese korrigiert sind.

Eine weitere Problematik taucht bei der Testbarkeit von Arrays auf, deren Pattern direkt von SSI/MSI-Anordnungen übernommen wurden. Die elegantesten Entwürfe mit der 74-TTL-Familie erweisen sich in monolithischer Chipform oft als nicht testfähig. Logische Fehler auf der Printplatte, wie Laufzeitunterschiede, Schwingverhalten, metastabile Zustände usw., konnten über Testpunkte am Board erkannt und meist noch beispielsweise mit Verzögerungsschaltungen oder ähnlichem behoben werden; in einem Gate-Array ist das nicht mehr möglich.

Dazu kommt noch, daß der Testingenieur des Herstellers und der des Anwenders von Arrays mit der jeweiligen Applikation genauso vertraut sein müssen wie der Entwickler des Bausteins, um das spezielle Testprogramm für diesen Baustein zu erstellen.

Die Einmaligkeit einer kundenspezifischen Array-Schaltung für eine ganz spezifische Applikation macht diese natürlich auch für einen Zweitlieferanten relativ uninteressant. Die Individualität der jeweiligen Entwicklung erschwert auch die nötige Applikationsunterstützung in Form von Software, Entwicklungssystemen, Anwendungshinweisen und Schaltungsrichtlinien.

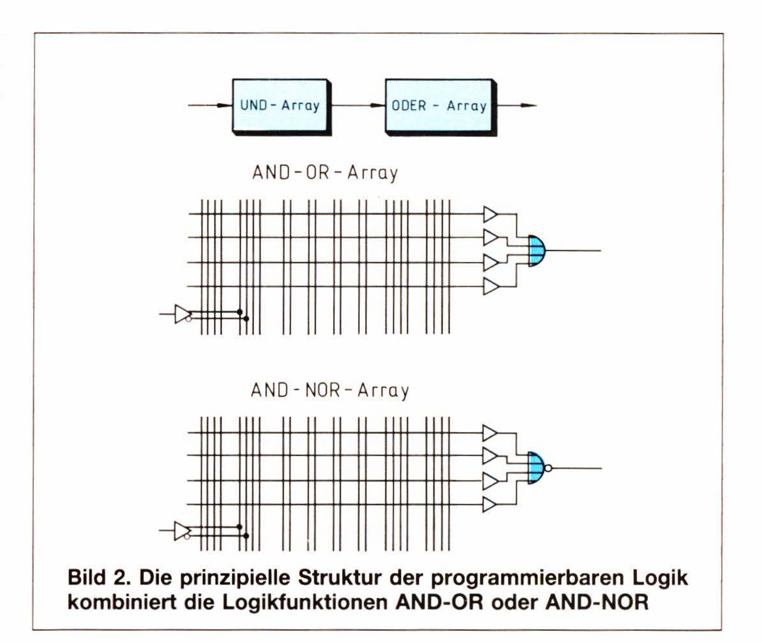


Bild 2. Die prinzipielle Struktur der programmierbaren Logik kombiniert die Logikfunktionen AND-OR oder AND-NOR

Der Entwickler von Gate-Arrays muß seine Schaltung voll dokumentieren und gegebenenfalls Unterstützung für den Systemingenieur geben, damit dieser das Bauelement richtig einsetzt.

Bauelemente auf der Basis von Gate-Arrays weisen eine verhältnismäßig geringe logische Dichte auf. Sie bestehen aus losen Strukturen mit großen Zwischenräumen für die Metallverbindungen. Für eine bestimmte Schaltung benötigt eine Array-Struktur für die gleiche Gatterzahl die zwei- bis fünffache Siliziumfläche einer Standardschaltung.

Programmierbare Logik (PAL)

Da Vor- und Nachteile kundenspezifischer Logik stets von der Stückzahl abhängig sind, stellt diese Abkehr von konventioneller TTL-Technik vor allem für kleine und mittlere Betriebe ein beträchtliches Unternehmer-

PAL ist...

Entdecken Sie Ihren „Goldfinger" beim Logik-Design: Die Semicustom-Lösung heißt PAL® von Monolithic Memories.

Hochkomplexe digitale Systeme erfordern heute Logik-Bauelemente, die den Anforderungen genügen: Flexibilität im Design, höchste Durchlaufgeschwindigkeiten, geringer Strombedarf, große Integrationsdichte. Programmierbar müssen sie sein – und genauso zuverlässig wie die Lieferbarkeit zu marktgerechten Konditionen.

PAL® (Programmable Array Logic) -Bauelemente von Monolithic Memories liegen hier goldrichtig. Das Spektrum des PAL-Erfinders umfaßt Bausteine in bipolar TTL, ECL und CMOS-Technologie; PALs mit anwenderorientierten Eigenschaften wie programmierbare Ausgangspolarität, Product Term Sharing, Bypass-Register, asynchrone Funktionen. Und reicht bis zur Familie der MegaPAL™ mit mehr als 5.000 frei programmierbaren Gatterfunktionen.

Für all diese Goldstücke gibt es auch die Design-Software: PALASM II™, das PAL-Assemblierprogramm.

® PAL ist eingetragenes Warenzeichen von Monolithic Memories Inc.
™ MegaPAL und PALASM II sind Warenzeichen von Monolithic Memories Inc.

Monolithic Memories GmbH
Mauerkircherstr. 4
8000 München 80
Telefon 089/984961



Logik in Gold.

Monolithic Memories MMI

risiko dar, weil die Rentabilität wegen der weit höheren Initialkosten erst bei einer größeren Anzahl verkaufter Systeme erreicht wird. Kundenschaltungen oder Semikundenschaltungen in der Form von Gate-Arrays sind teuer – zu teuer, wenn die Produktionsstückzahl zu klein ist. SSI/MSI-Logik ist billiger, und die Bauelemente sind in Einzelstückzahlen erhältlich. Sie brauchen aber viel Platz für wenige Funktionen. Programmierbare Logik stellt hier die Alternative dar.

Mit Gate-Arrays und programmierbarer Logik werden kundenspezifische hochintegrierte Schaltungen hergestellt. Es lassen sich mit beiden Konzepten gleiche logische Funktionen erfüllen. Ein wesentlicher Unterschied besteht darin, daß ein PAL beim Anwender mit einem konventionellen PROM- bzw. PAL-Programmiergerät auf seine Funktionsweise festgelegt wird, während beim Gate-Array der Halbleiterhersteller den letzten Fertigungsschritt macht.

Bei PALs handelt es sich also um programmierbare Gate-Arrays.

Dabei wird aber die fertigungsbedingte Notwendigkeit, eine Standardschaltung zu liefern, mit der hohen Flexibilität von Semi-Kundenschaltungen verbunden.

Mit PALs lassen sich viele der früher mühsam mit den SSI/MSI-Familien aufgebauten Schaltungen realisieren. Sie können jedoch den spezifischen Anforderungen bestimmter Applikationen angepaßt werden. Diese Anpassung erfolgt über die Programmierung durch den Anwender; die Durchlaufzeiten zu den ersten Musterbausteinen und auch zur Produktion betragen oft nur wenige Minuten.

Diese Bausteine bieten dem Entwickler unbegrenzten Spielraum, seine Ideen sofort auf Silizium zu bringen. Daraus ergibt sich eine wesentliche Verkürzung der Entwicklungszeit mit der damit verbundenen größtmöglichen Flexibilität, niedrigsten Initialkosten, Testzeiten und kompletter Dokumentation im Hause des Anwenders. Handelsübliche Mikroprozessorentwicklungssysteme oder Personal Computer lassen sich hierzu effektiv einsetzen.

Wurden PAL-Bauelemente bisher nur als Gatterersatz betrachtet, so ermöglichen es neue und weiterentwikkelte Elemente mit einer Komplexität von bis zu 6000 Gatterfunktionen, auch „intelligente“ eigene Funktionen zu definieren. Diese Bausteine sind bereits voll verfügbar und ebenfalls frei programmierbar.

Was ist programmierbare Logik?

Der erste Blick auf ein PAL-Logikdiagramm ist sicher verwirrend. Es zeigt ein Gatterfeld mit Ein- und Ausgängen, eine UND/ODER-Anordnung mit einer bestimmten Anzahl von Produktterme, Registerfunktionen und Ausgänge mit Rückkopplungen.

Im unprogrammierten Zustand ist alles mit allem verbunden. Durch Trennen einiger dieser Verbindungsstellen wird dem Baustein die jeweilige Funktion gegeben. Diese Programmierung kann durch Schmelzen von Sicherungen („Fusible-Link“) oder Aufladen von Ladungsträgern (MOS) erfolgen.

Die Auswahlkriterien und Schlüsselparameter für die Beschreibung programmierbarer Logik sind:

- ○ Anzahl der Ein- und Ausgänge
- ○ Anzahl der Produktterme
- ○ Registerfunktionen
- ○ Technologie (Durchlaufgeschwindigkeit, Stromverbrauch)
- ○ Applikationsunterstützung (Software, Literatur usw.)

Jede Ausgangsfunktion besteht aus der Summe (ODER-Verknüpfung) einer bestimmten Anzahl von Produkten, wobei jedes Produkt (UND-Verknüpfung) aus den jeweiligen Polaritäten der Eingänge gebildet wird.

Bild 2 zeigt die bekannte UND/ODER-Anordnung, die als Funktion in vielen SSI/MSI-Elementen verwendet

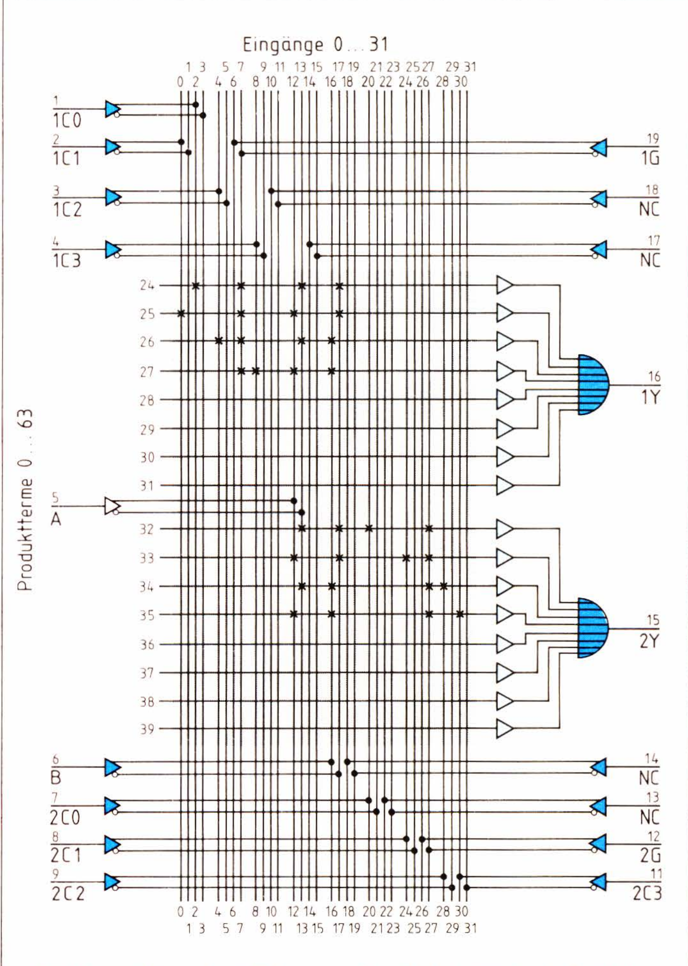


Bild 3. Mit Hilfe eines PALs vom Typ 16H2 simulierte Funktion des TTL-Bausteins 74153. Die eingezeichneten Kreuzungspunkte sind durch die Programmierung geöffnet worden

wird. Nehmen wir zum Beispiel den 74153, einen zweifachen Multiplexer mit je vier Eingängen. Diese Funktion kann in einer programmierbaren Logikstruktur entsprechend *Bild 3* erzeugt werden.

Die gängige Beschreibung von Hardware ist der logische Schaltplan, und zum Testen dieser Schaltung wird meist ein Prototyp verwendet. Beide Methoden sind aber unzulängliche Werkzeuge für die Erstellung von LSI-Schaltkreisen und dementsprechenden zeitkritischen Schaltungen. Ein logisches Schaltbild mag für menschliche Begriffe verständlich sein, eignet sich aber in den wenigsten Fällen für die Eingabe in einen Rechner.

Desgleichen stellt auch der Prototyp einen ungeschickten Versuch dar, eine Schaltungsentwicklung zu testen und zu verifizieren. Unterschiede in den Durchlaufzeiten, der Schaltungsausführung sowie des Logikkonzepts beeinträchtigen den Wirkungsgrad der mit Schaltplan und Prototyp entwickelten Schaltung. PAL-Bausteine werden aber nicht, wie sonst bei Logik üblich, mit Schaltbildern definiert, sondern über logische Gleichungen, wie dies an Hand typischer Logikschaltungen in *Bild 4* skizziert ist.

Während eines Entwicklungsprozesses wäre sehr oft eine zusätzliche Eigenschaft notwendig, um einfacher zu einer Problemlösung zu kommen. Zum Beispiel wäre es oft sinnvoll, einen 3-zu-8-Multiplexer mit „active high" Ausgängen zu haben. Bei Verwendung von PALs kann nunmehr jede gewünschte Funktion mit Hilfe von Boole'scher Algebra frei programmiert und auch jederzeit leicht verändert (editiert) werden.

Die direkte Logikerstellung mit Hilfe logischer Gleichungen macht die Systementwicklung mit PALs sehr einfach. Änderungen der Funktion sind durch bloßes Austauschen der PALs möglich.

Wesentliche Merkmale programmierbarer Logik

Kompatibilität und Optimierung von Schaltungen und Platzersparnis

PALs mit einer Komplexität zwischen 300 und 6000 Gatterfunktionen integrieren eine Reihe herkömmlicher Logikbauelemente in einem Gehäuse. Einen Ersetzungsgrad anzugeben, ist nicht sinnvoll, da dieser sehr stark von der jeweiligen Applikation abhängt, sie ermöglichen jedoch oft Schaltungen, die in den herkömmlichen Logikreihen einfach nicht vorkommen.

Mit der Reduktion der Anzahl der Bauelemente und der damit verbundenen Vereinfachung des Platinenlayouts lassen sich die Leiterplatten wesentlich kompakter aufbauen. In Mehrplatinensystemen besteht darüber hinaus die Möglichkeit, einige Karten einzusparen, so daß verschiedene Steckerverbindungen und Verdrahtungsarbeiten ebenfalls wegfallen.

Platzersparnis resultiert in niedrigeren Systemkosten oder ermöglicht es, mit gleichem Aufwand mehr Funktionen auf gleichem Platz unterzubringen.

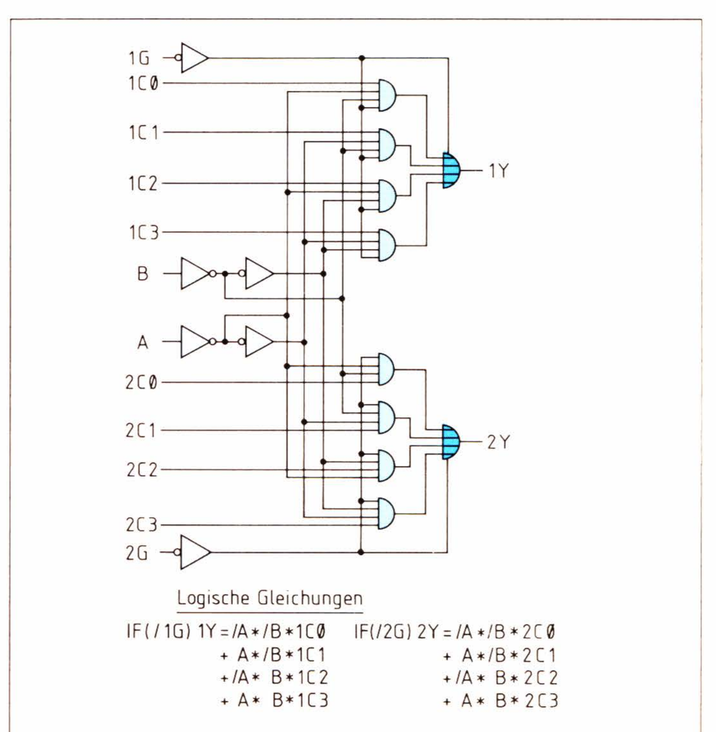


Bild 4. Typische Standard-Logikschaltungen lassen sich durch Boole'sche Algebra beschreiben. Hier als Beispiel ein Multiplexer 4-auf-1

Freie Wahl der Pinbelegung

Die Realisierung logischer Schaltungen bringt schon bei geringer Gatterzahl ein großes Problem der Leiterbahnführung beim Platinenlayout mit sich. Drahtbrükken oder Durchkontaktierungen bei zweiseitigem Layout, die in der Serienfertigung die Montagekosten wesentlich erhöhen, sind die Folge. PAL-Bausteine reduzieren nicht nur die Anzahl der Leiterbahnen, sondern bieten auch noch einen weiteren Vorteil, der bei zunehmend rechnergestützter Layout-Erstellung an Bedeutung gewinnt:

Die Ein- und Ausgänge der PALs sind frei wählbar.

Durch geschickte Pinbelegung der PAL-ICs während der Rechnersynthese ist es möglich, die Leiterbahnen bereits auf dem Chip zu entflechten. Die vom Hersteller bestimmte Pinbelegung bei Standard-Logik kann somit durch die flexibel programmierbare Pinbelegung ersetzt werden. Unbenutzte Eingänge müssen auch nicht mehr, wie bei TTL-Bausteinen, auf einen bestimmten logischen Pegel festgelegt werden, sondern bleiben unbeschaltet; der Baustein selbst sorgt dann mit seiner inneren Beschaltung für die Fixierung dieses Pegels.

Flexibilität

PAL-Bauelemente unterscheiden sich von allen Logikkonzepten u. a. dadurch, daß sie am einfachsten Änderungen zulassen. Sie sind die einzigen Baule-

mente, mit denen während der Entwicklungsphase und Produktion bei bereits fertiggestellten Leiterplatten Änderungen in der Hardware möglich sind.

Geräte können mit Hilfe von PALs an die verschiedensten Märkte und Marktnischen angepaßt werden.

Da auch Kundenoptionen mit Hilfe dieser flexiblen Logik leicht möglich sind, eröffnen sich dynamischen Unternehmen vielfältige Chancen, Marktlücken durch neue Produkte kostengünstig, konkurrenzfähig und schnell abzudecken.

Schnelle Entwicklungszeiten

Programmierbare Logik gewährleistet eine drastische Verringerung von Entwicklungszeiten. Beim Erstellen des Blockschaltbildes wird an Stelle der logischen Funktion ein PAL-Bauelement gesetzt. Die spezielle Architektur sowie das Interface zwischen den logischen Blöcken kann dann später mit minimalem Aufwand editiert werden.

Jede Adaption als Folge logischer Fehler oder sonstiger Änderungen in den Spezifikationen des Gerätes läßt sich durch Umprogrammierung von PALs leicht durchführen.

Hohe Schaltgeschwindigkeiten

Typische Laufzeiten von Signalen zwischen Ein- und Ausgängen liegen bei PAL-Bausteinen heute bei rund 10 ns. Bei PALs werden zusätzlich die Eingangssignale parallel verarbeitet.

PALs sind effektiv wesentlich schneller als TTL-Schaltungen.

Noch dazu ermöglicht programmierbare Logik homogene Durchlaufzeiten; verschiedene Datenpfade innerhalb eines Bauelements weisen zueinander keine Laufzeitunterschiede auf.

Ing. (grad.) Willy Voldan ist gebürtiger Wiener. Er studierte am dortigen TGM Nachrichtentechnik und Elektronik. Seit 1979 ist er bei Monolithic Memories in München als Applikationsingenieur tätig, wobei eines seiner Hauptaufgabengebiete im Bereich der programmierbaren Logik liegt.



Datenschutz

Insbesondere für die mittelständische Industrie ist der Kopierschutz eines Projektes ein mitunter lebenswichtiger Faktor. Schwierigkeiten können hierbei immer dann auftreten, wenn kundenspezifische Bausteine bei Halbleiter-Herstellern in Auftrag gegeben oder fertige Schaltungen von Konkurrenzunternehmen kopiert werden. Dieses Problem ist nur durch Schaltungsentwürfe zu lösen, die rein software-orientiert sind.
Mit PALs besteht die Möglichkeit, zwei zusätzliche Sicherungen im Bauelement zu trennen.

Die Duplikation eines PALs in einem Programmer ist nicht mehr möglich.

Mit vielen Second-Source-Lieferanten, stark reduzierten Preisen, verschiedenen Technologien und einer Vielzahl verschiedener Konfigurationen und Typen sind PALs zu wesentlichen Bestandteilen in jedem digitalen System geworden. Damit auch die nächste Generation programmierbarer Logik erfolgreich eingesetzt werden kann, sollten diese Bauelemente von Beginn der Systementwicklung an bereits in Betracht gezogen werden. Das erfordert allerdings Vertrauen zwischen Hersteller und Anwender.

Willibald Voldan

Die Architektur programmierbarer Logik

Programmierbare Logikbausteine (PAL und PLA) haben vieles mit PROMs gemeinsam. Alle drei Bauelementefamilien sind intern aus den gleichen logischen Strukturen (UND-ODER) aufgebaut. Aufgrund der jeweiligen Anordnung der programmierbaren Arrays unterscheiden sich diese drei Familien voneinander jedoch in den logischen Eigenschaften. Da hierdurch auch die Anwendungsbereiche beeinflußt werden, soll dieser Beitrag die verschiedenen Strukturen programmierbarer Logikbausteine erläutern.

Bild 1 zeigt die grundsätzliche UND-ODER-Struktur programmierbarer Bauelemente. Diese zweistufige Anordnung besteht aus der Eingangs-UND-Matrix, in der die logischen Produkte der verschiedenen Eingänge gebildet werden. Das nachfolgende ODER-Array faßt die UND-Verknüpfungen zu den jeweiligen Ausgängen zusammen und bildet die für Boole'sche Algebra wichtige UND-ODER-Konfiguration. Diese Anordnung entsteht ebenfalls bei der Logik-Erstellung mit Karnaugh-Diagramm oder mit dem Minimierverfahren nach McCluskey.

Im *Bild 2* ist eine UND-Verknüpfung in der herkömmlichen Darstellungsform und in der Zeichnungsweise von Array-Schaltungen dargestellt. Alle Eingänge zu dieser Matrix, invertierend und nicht-invertierend, sind dabei mit jedem UND-Gatter dieser Matrix verbunden. Daraus folgt, daß innerhalb von programmierbaren Bauelementen jeder Eingang zu jedem beliebigen Ausgang logisch verknüpft werden kann. Diese sehr wesentliche Eigenschaft nennt man die „allumfassende Verknüpfbarkeit" eines Bauelementes. Damit unterscheiden sich programmierbare Bausteine ganz wesentlich von Produkten der Standard-Logik-Familien.

PROM-Architektur

Bild 3 zeigt die Konfiguration eines PROMs mit 4 Eingängen, 16 Speicherzellen und 4 Ausgängen. Sie besteht aus einem festgelegten Eingangs-UND-Array, das ein nachfolgendes programmierbares ODER-Array treibt. Die wesentliche Eigenschaft von PROMs liegt darin, daß die Eingänge (Adressen) von dem nicht programmierbaren UND-Array komplett decodiert sind.

Jede Kombination der Eingänge wird von einem bestimmten UND-Gatter dargestellt. Die Anordnung mit einer Anzahl von N Eingängen mit 2^N Kombinationsmöglichkeiten erfordert demnach 2^N UND-Gatter. Die Schaltung von Bild 3 besitzt daher 16 UND-Gatterfunktionen in der Eingangsstufe.

Durch die Programmierung des ODER-Feldes für den jeweiligen Ausgang kann das PROM jede logische Funktion erfüllen und ist damit nur durch die Anzahl der Ein- und Ausgänge limitiert. Für jeden Ausgang des Bausteins ist eine eigene und von den anderen Ausgängen unabhängige logische Funktion programmierbar.

Ein gewaltiger Nachteil der PROM-Konfiguration für die Anwendung in Logik-Schaltungen besteht in der eingeschränkten Verfügbarkeit von Ein- und Ausgängen.

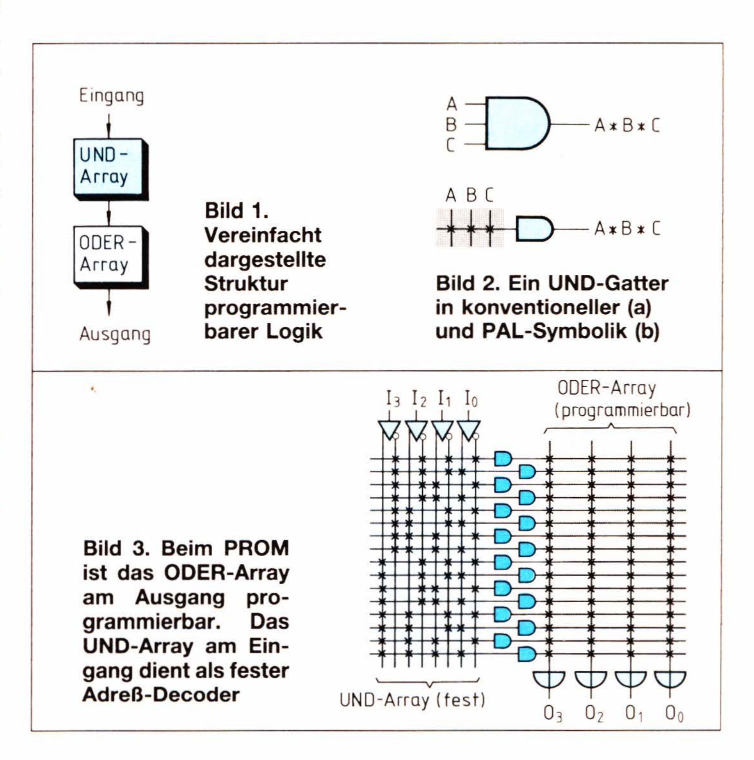


Bild 1. Vereinfacht dargestellte Struktur programmierbarer Logik

Bild 2. Ein UND-Gatter in konventioneller (a) und PAL-Symbolik (b)

Bild 3. Beim PROM ist das ODER-Array am Ausgang programmierbar. Das UND-Array am Eingang dient als fester Adreß-Decoder

Jeder zusätzliche Eingang verdoppelt die Anzahl der notwendigen UND-Gatter und damit auch die Größe der programmierbaren und festgelegten Felder. So besitzt z. B. ein PROM mit 1024 × 4 Bit 10 Eingänge und 4 Ausgänge. Benötigt die jeweilige Logik nun aber 11 Eingänge, muß bereits ein doppelt so großes PROM (2048 × 4) verwendet werden.

Logische Funktionen bestehen aber nicht immer aus einer festen Anzahl von Ein- und Ausgängen. Typische Logikanordnungen können schnell 16 Eingänge und mehr aufweisen. Nur 16 Eingänge würden aber bereits ein PROM mit 64 K × 8 Bit erforderlich machen.

Bild 4. Das PAL stellt sozusagen ein spiegelverkehrtes Abbild des PROMs dar

Die oben beschriebene Architektur und deren Eigenschaften gelten für jede Speicherkonfiguration wie PROM, EPROM, EEPROM und RAM.

PAL-Architektur

Die interne Struktur eines PAL-Bausteins (*Bild* 4) ist das exakte Gegenteil der PROM-Konfiguration. Zwar besteht auch das PAL aus zwei voneinander getrennten Arrays, dabei ist aber bei diesen Bausteinen das Eingangs-UND-Feld programmierbar und das Ausgangs-ODER-Array festgelegt. Diese Anordnung beseitigt die Grenzen der PROM-Architektur hinsichtlich der beschränkten Verfügbarkeit von Ein- und Ausgängen. Ein PAL-Baustein kann beliebig viele Ein- und Ausgänge besitzen. Die Begrenzung liegt nur in der Größe des Gehäuses. Jeder zusätzliche Eingang vergrößert das programmierbare Array nur unwesentlich.

Ein PAL-Baustein mit N Eingangsanschlüssen besitzt 2 ∗ N Eingänge zu jedem UND-Gatter im programmierbaren Array. Das PAL10L8 mit 10 Eingängen und 8 Ausgängen besitzt daher 20 Eingänge zu jeder der UND-Funktionen. In der programmierbaren Matrix steht jeder Schnittpunkt von Eingangsschienen mit Produkttermen für eine Verbindung eines Eingangs mit dieser UND-Funktion. Diese Verbindung wird bei heutigen PALs über programmierbare Sicherungen realisiert.

Die festgelegte ODER-Matrix bestimmt, welche UND-Funktionen zu welchem Ausgang geführt werden. *Bild* 5 zeigt einen Ausschnitt einer PAL-Innenschaltung, bei der immer zwei Produkt-Terme (UND-Funktionen) pro ODER-Funktion zum Ausgang geführt sind.

In der festen Verdrahtung der Produkt-Terme zu den jeweiligen ODER-Gattern liegt der Nachteil der PAL-Architektur – die Anzahl der für eine Schaltung benötigten Produkt-Terme darf die im Baustein vorhandenen nicht überschreiten. Trotzdem ermöglicht es nur diese Konfiguration, wichtige architektonische Eigenschaften mitzuintegrieren, die PALs zu geradezu idealen Bauelementen für die Realisierung komplexer logischer Schaltungen machen.

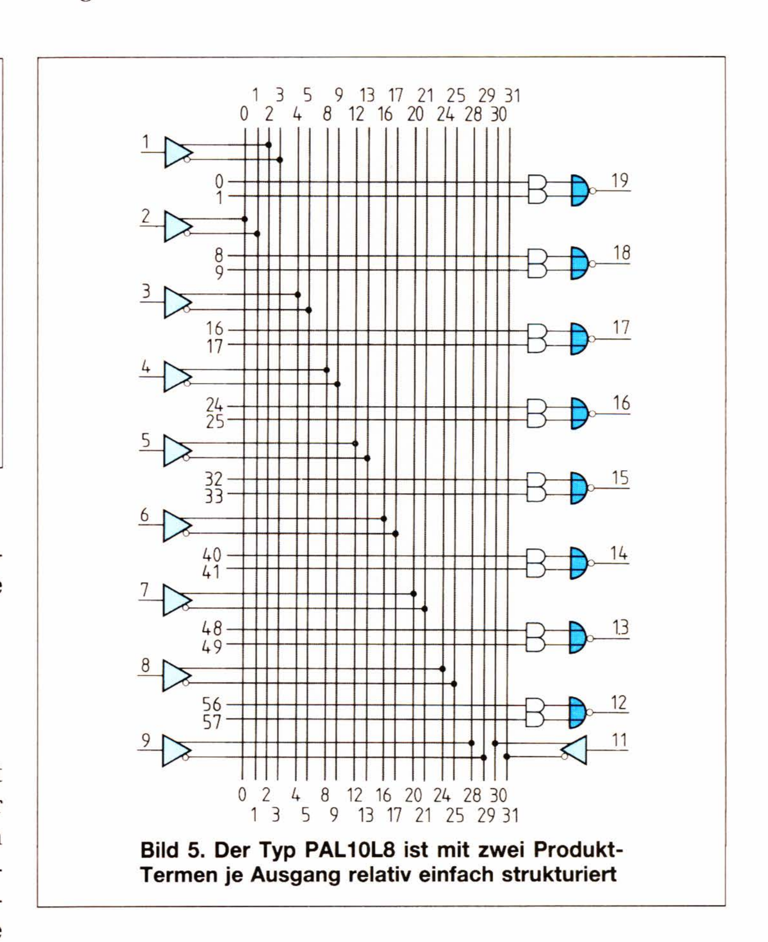


Bild 5. Der Typ PAL10L8 ist mit zwei Produkt-Termen je Ausgang relativ einfach strukturiert

PLA-Architektur

Die Kombination der PROM- mit der PAL-Architektur ergibt die Struktur einer PLA-Schaltung. Dabei ist sowohl das UND-Array als auch das ODER-Array programmierbar. Mit dieser Konfiguration ist die Anzahl der UND-Funktionen pro Ausgang nun nicht mehr festgelegt, sondern variabel programmierbar. Das ergibt in einigen Anwendungsfällen eine größere Flexibilität der Logikschaltung.

Der Nachteil der PLA-Struktur liegt vornehmlich im kommerziellen Bereich. Die Durchlaufzeit und der größte Teil der Verlustleistung geht im programmierbaren Array verloren. Zwei programmierbare Felder ergeben in jedem Fall höhere Durchlaufzeiten und verbrauchen mehr Versorgungsstrom als nur ein programmier-

bares Feld. Eine größere Chipfläche der PLA-Struktur resultiert darüber hinaus auch in höheren Herstellungskosten.

Die verschiedenen PAL-Konfigurationen

Mit der oben beschriebenen PAL-Konfiguration können zunächst sämtliche Grundgatter wie AND, NAND, OR, NOR, INVERTER, EX-OR und EX-NOR realisiert werden. Weiterhin erlaubt dieses Konzept dem Anwender, die Ein- und Ausgänge des Bausteins beliebig festzulegen und damit die Entflechtung bereits auf dem Chip durchzuführen.

Die Gatterkomplexität reicht derzeit bis über 5000 frei programmierbare Gatterfunktionen. Eine Verdichtung von Logikbausteinen wird bereits bei kleinen PALs mit mindestens 5:1 erreicht, kann aber je nach Applikation auch wesentlich höher sein (bis weit über 30:1).

Um jeder Anforderung gerecht zu werden, d. h. Flexibilität mit größtmöglicher Integration von logischen Funktionen zu verbinden, wurden mehrere Familien von PAL-Bausteinen entwickelt, die diese Bedingungen erfüllen. PAL-Bausteine im 20- und 24poligen Gehäuse sind von 12 × 10 (12 Eingänge, 10 Ausgänge) bis 20 × 2 in jeder Konfiguration mit „active high“, „active low“ und auch programmierbarer Ausgangspolarität verfügbar. PALs im 40- und 84poligen Gehäuse sind in einer Konfiguration 32 × 16 und 64 × 32 verfügbar (Mega-PAL).

Kombinatorische Logik

PALs für ausschließlich kombinatorische Logik bestehen aus der oben beschriebenen Anordnung von UND-ODER-Arrays.

Je nach PAL-Typ ist eine unterschiedliche Anzahl von UND-Verknüpfungen pro Ausgang vorhanden. Bei der im *Bild 6* gezeigten Logikzelle eines PAL12L6 führen vier Produktterme über das NOR-Gatter (active low) zum Ausgang.

Kombinatorische PAL-Bausteine eignen sich hervorragend zur Herstellung von Chip-Select-, Decoder- und Multiplexer-Funktionen.

Kombinatorische Logik mit Rückkopplung

Im *Bild 7* ist eine bidirektionale Logikzelle eines PAL-Bausteins dargestellt. Diese Struktur weist nicht nur mehr Produktterme als ein 12L6 auf, sie ermöglicht es auch, den jeweiligen Pin sowohl als Ausgang als auch als Eingang zu programmieren. Diese Eigenschaft ist außerordentlich wichtig für programmierbare Logik, erlaubt sie doch die Anpassung der erforderlichen Anzahl der Ein- und Ausgänge an die jeweilige Anwendung.

Eine besondere Eigenschaft dieser bidirektionalen Struktur ist die Möglichkeit, die Freigabeschaltung (ENABLE) der Ausgänge als Funktion bestimmter Eingänge im Array zu programmieren, da diese Verknüpfung ebenfalls über einen Produktterm geführt wird.

Die im Bild 7 gezeigte PAL-Struktur kann in einem der drei folgenden Funktionsmodi betrieben werden:

- Ausschließlich als Ausgang mit interner Rückkopplung des Signalweges;
- Ausschließlich als Eingang;
- Dynamisch steuerbare Three-State-Funktion (Ein- und Ausgang).

Werden alle Sicherungen im obersten Produktterm der Logikzelle getrennt, so liegt das zugehörige UND-Gatter immer auf V_{CC}. Die dementsprechende Steuerleitung gibt daher den Ausgangstreiber frei und schaltet den Anschluß dauernd als Ausgang. Das Signal kann über die interne Rückkopplung, invertiert oder nicht, wieder in das Array zurückgeführt werden. Dieser Rückkopplungspfad erlaubt mit dem Weiterschleifen der Signale die Realisierung tieferer Logik durch die mehr-

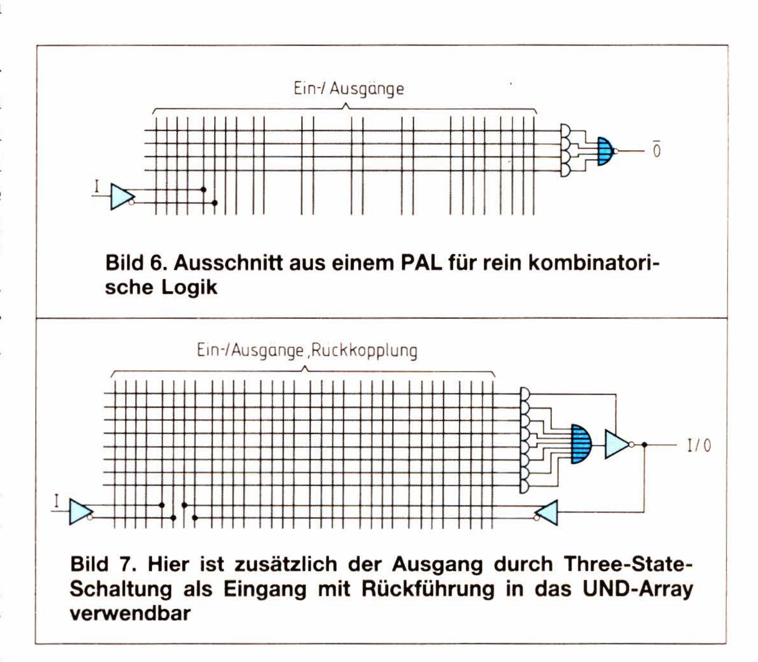


Bild 6. Ausschnitt aus einem PAL für rein kombinatorische Logik

Bild 7. Hier ist zusätzlich der Ausgang durch Three-State-Schaltung als Eingang mit Rückführung in das UND-Array verwendbar

malige Verwendung der UND-ODER-Struktur. Dabei erhöht sich natürlich auch jeweils die Durchlaufzeit.

Soll der Anschluß „I/O“ ausschließlich als Eingang verwendet werden, bleiben alle Sicherungen im obersten Produktterm dieser Logikzelle bestehen, das dementsprechende UND-Gatter ist daher immer inaktiv und schaltet den Ausgangstreiber hochohmig. Die restlichen sieben Produktterme dieser Logikzelle werden bei dieser Schaltungsart nicht mehr verwendet, es wurde aber ein zusätzlicher Eingang gewonnen. Damit ist der Entwickler der Schaltung nicht an eine vorgegebene Anzahl von Ein- und Ausgangskombinationen gebunden. Er kann das Bauelement wie in einem Puzzle für die jeweils vorliegende Problemstellung anfertigen.

Die dritte Möglichkeit sieht vor, den Ausgang dynamisch steuerbar als Ein-/Ausgangstreiber über eine logische Verknüpfung eines oder mehrerer Eingänge zu beschalten. Der Vorteil dieser Beschaltung liegt darin, daß mit dieser Schaltungsart trotz eines zusätzlichen Eingangs nicht das übrige vorgeschaltete programmier-

bare Array dieses Ausgangs verlorengeht. Diese Anwendung ist besonders in Steuerungen bzw. Bus-Systemen außerordentlich wichtig.

Ein serieller Ein-/Ausgangs-Anschluß ist typisch in einer Schieberegisterfunktion. Werden z. B. Daten nach links verschoben, funktioniert der Anschluß als Eingang, beim Rotieren nach rechts wird er zum Ausgang.

Registerausgänge mit Rückkopplung

Um nicht nur kombinatorische, sondern auch sequentielle Schaltungen realisieren zu können, besitzen einige PAL-Bausteine Registerfunktionen (*Bild 8*). Die Grundstruktur dieser Bauelemente ist gleich der von kombinatorischen PALs, nur werden die Signale aus der UND-ODER-Struktur in den D-Eingang eines Registers geführt. Der Komplementär-Ausgang dieses Flipflops ist dann wieder in die programmierbare UND-Matrix rückgekoppelt. Damit kann sich der Baustein an den jeweiligen Zustand „erinnern" und die nächste Übergangsfunktion von der vorhergehenden ableiten. Die Ausgangstreiber werden von einem für alle Registerausgänge gemeinsamen ENABLE-Eingang gesteuert.

Im Gegensatz zu der oben beschriebenen bidirektionalen Zelle wird bei der Register-Architektur die Rückkopplung vom Flipflop, also vor dem Ausgangstreiber, geführt. Diese Konfiguration ist sinnvoll, weil damit die Zustandsinformation zu jedem Zeitpunkt verfügbar ist und auch nach einem Three-State-Schalten der Ausgänge nicht verlorengeht. Dies erleichtert die Entwicklung von Zustandsmaschinen, wie z. B. Zähler und Schieberegisterfunktionen, ganz wesentlich. Die Register-PALs können u. a. zur Datenspeicherung, zur Datenweiterschaltung (sequentielle Datenverarbeitung) und für Steuerungsaufgaben programmiert werden. Das Resultat wird dann nach der Freigabe des Enable auf die Ausgänge geschaltet.

Register-PALs mit Exklusiv-ODER-Funktionen

Die Mitglieder dieser PAL-Familie besitzen noch zusätzlich zu der herkömmlichen Struktur mitintegrierte Exklusiv-ODER-Funktionen (*Bild 9*).

Die UND-Gatter pro Ausgang sind in zwei Gruppen unterteilt und über ein Exklusiv-ODER-Gatter am D-Eingang des Registers geführt. Das zusätzliche EXOR-Gatter ermöglicht eine einfache Realisierung der „HALTE"-Funktion in Zählern oder anderen Zustandsmaschinen. Es läßt sich auch zur dynamischen Steuerung der Ausgangspolarität verwenden.

PALs für arithmetische Anwendungen

Bei einigen PAL-Bauelementen wurde der simple Eingangstreiber durch einen Halbaddierer ersetzt. Zusammen mit dem Register, das in diesen Bausteinen auch als Akkumulator bezeichnet werden kann, funktioniert diese Anwendung als Rechenwerk für einfache arithmetische Operationen wie Addieren, Subtrahieren und Vergleichen (*Bild 10*).

Die Exklusiv-ODER-Funktion am Eingang des Flipflops erlaubt es, Übertragssignale von vorausgegangenen Operationen mit einer logischen Summe aus dem programmierbaren Array zu vergleichen. Der Register-Ausgang ist wieder in die Matrix rückgekoppelt und ermöglicht damit sämtliche Boole'sche Funktionen, die im Karnaugh-Diagramm (*Bild 11*) dargestellt sind.

Alle Three-State-Ausgänge der PAL-Bausteine liefern bis zu 24 mA Treiberstrom; in Bus-Applikationen kann man also noch zusätzliche Treiber-Bauelemente einsparen.

Das erste Grundkonzept der PAL-Familie sah nur ein programmierbares Array, festgelegte Ausgangspolarität (entweder „High"- oder „Low"-Active) und synchrone Registerfunktionen vor, um ein hohes Fertigungsvolumen (Verfügbarkeit hoher Stückzahlen), beste elektrische Spezifikationen, einfachste Programmierung und damit eine leichte Handhabung zu gewährleisten. Heute ist es möglich, über weiter verbesserte Fertigungseinrichtungen, kleinere Geometrien sowie neue Technologien diese programmierbare

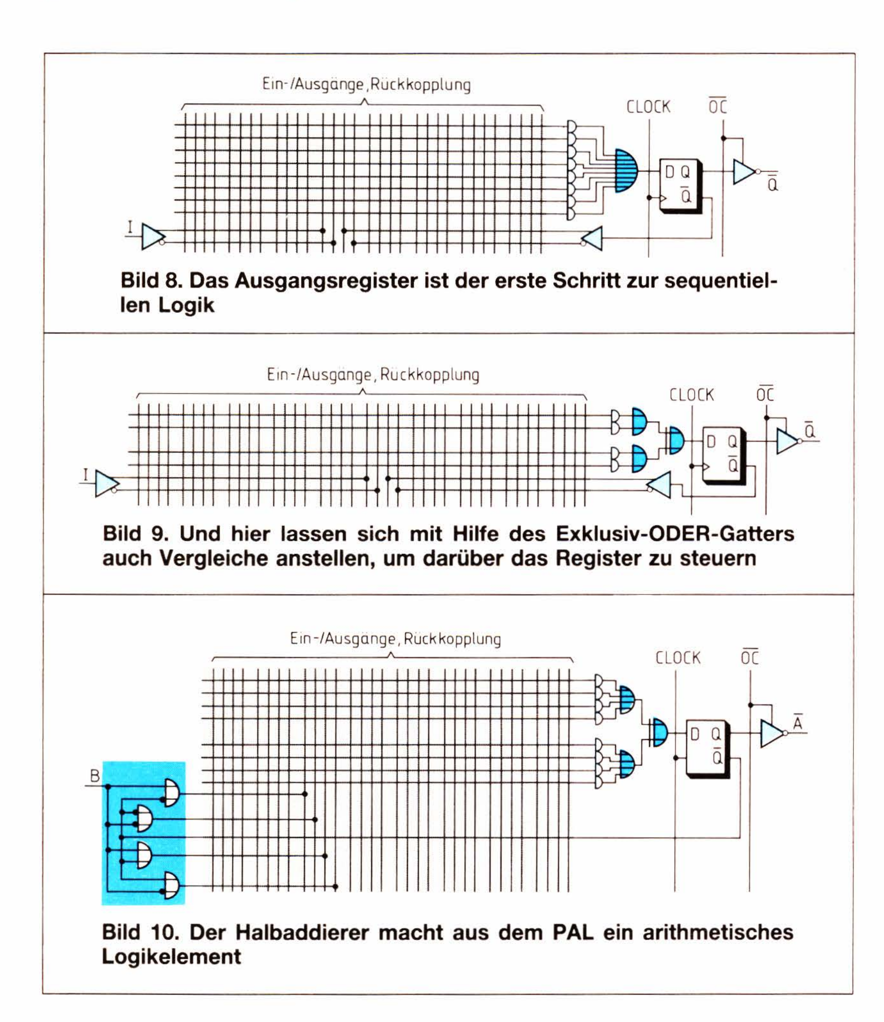


Bild 8. Das Ausgangsregister ist der erste Schritt zur sequentiellen Logik

Bild 9. Und hier lassen sich mit Hilfe des Exklusiv-ODER-Gatters auch Vergleiche anstellen, um darüber das Register zu steuern

Bild 10. Der Halbaddierer macht aus dem PAL ein arithmetisches Logikelement

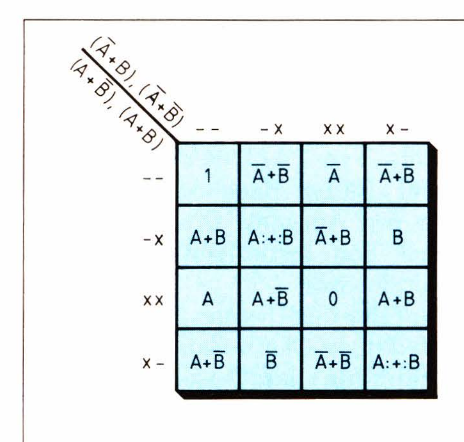


Bild 11. Was man mit einem PAL nach Bild 10 anfangen kann, zeigt das Karnaugh-Diagramm

Logik-Familie in ihren Funktionen und Anwendungen wesentlich zu erweitern.

Neueste Entwicklungen

Die Parameter zur Beschreibung programmierbarer Logik sind:
- Anzahl der Eingänge;
- Anzahl der Ausgänge;
- programmierbare logische Funktionen;
- Anzahl der Produktterme;
- Signal-Durchlaufzeit;
- Stromverbrauch.

PALs mit programmierbarer Ausgangspolarität

War es mit den herkömmlichen PAL-Bausteinen je nach Type nur möglich, entweder „High"- oder „Low"-aktiv-Ausgänge zu definieren, erlaubt die Integration eines zusätzlichen Exklusiv-ODER-Gatters, die Polarität jedes Ausgangs des Bausteins individuell zu programmieren (*Bild 12* und *13*).

Nach der UND-ODER-Anordnung wird das Signal über ein Exklusiv-ODER-Gatter geführt, dessen zweiter Eingang über eine Sicherung an Masse angeschlossen ist. Bleibt diese Sicherung bestehen, durchläuft das jeweilige Signal das XOR-Gatter unverändert. Nach der Programmierung dieser Sicherung wird das XOR zum Inverter und das Signal als logisches Komplement an den Ausgang geführt.

Mit der programmierbaren Ausgangspolarität werden bei manchen Anwendungen sicher auch noch Produktterme eingespart, weil die lästige Umwandlung nach den DeMorgan'schen Gesetzen entfällt.

PALs mit „Product-Term-Sharing"

Die Architektur programmierbarer Logikbausteine besteht im wesentlichen aus zwei Arrays, nämlich den UND-ODER-Matrizen. Die Eingänge zum UND-Array können zu jedem der vorhandenen UND-Gatter beliebig oft ein Produkt bilden. Daher nennt man diese Anordnung der UND-Gatter auch Produkt-Terme. Das logische Produkt wird dann über ein festgelegtes bzw. programmierbares ODER-Array an den Ausgang geführt. Die Anzahl der Produkt-Terme pro Ausgang bestimmt daher maßgeblich die logische Tiefe (ODER-Verknüpfungen) eines PAL-Bauelementes. Die Summe der Produkt-Terme (PT) in einem Bauelement und damit auch pro Ausgang kann aber nicht beliebig erhöht werden, da sie maßgeblich am Leistungsverbrauch des Bauelementes beteiligt ist (mehr PT ergeben fast linear mehr Stromverbrauch).

Um nun trotzdem die Zahl der PT pro Ausgang erhöhen zu können, implementierte MMI in seinen neuesten PALs die Eigenschaft des „Product-Term-Sharings". Dabei werden über ein weiteres programmierbares Feld jetzt immer 16 PT (bisherige PALs nur 8) zwei nebeneinanderliegenden Ausgängen zugeordnet. Über Programmierung der Sicherungen in diesem Array kann nun jeder der beiden Ausgänge eine Anzahl zwischen 0 und 16 PT erhalten, also z. B. 1:15, 2:14, 8:8, 16:0 usw. (*Bild 14*). Das bedeutet, daß damit ein Produkt-Term entweder dem einen Ausgang oder dem anderen oder beiden oder auch keinem zugeordnet werden kann. Mit dieser Variation von PTs können wesentlich komplexere logische Gleichungen in diesen PALs untergebracht werden.

Wie oben beschrieben, ermöglicht es diese Architektur, einen oder mehrere PTs auch keinem der beiden Ausgänge zuzuordnen. Dabei werden eben beide Siche-

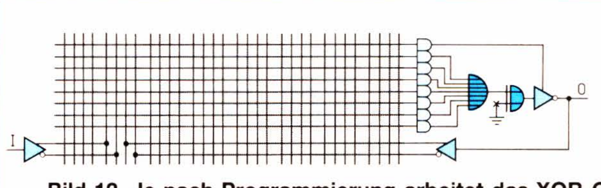


Bild 12. Je nach Programmierung arbeitet das XOR-Gatter als Inverter oder lediglich als Puffer. Das Resultat ist eine programmierbare Ausgangspolarität

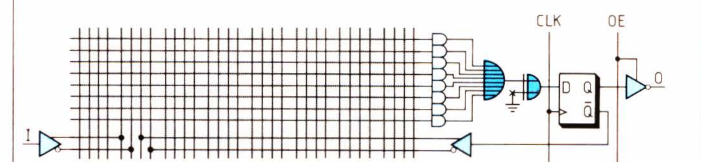


Bild 13. Sequentiell mit wählbarer Ausgangspolarität geht es in diesem PAL zu

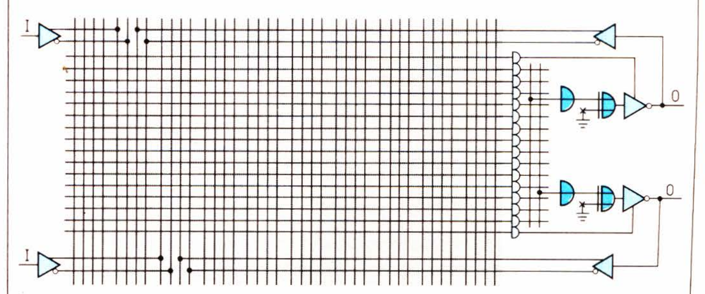


Bild 14. PAL mit „Product Term Sharing" und programmierbarer Polarität des Ausgangssignals. Hier für kombinatorische Logik ausgelegt

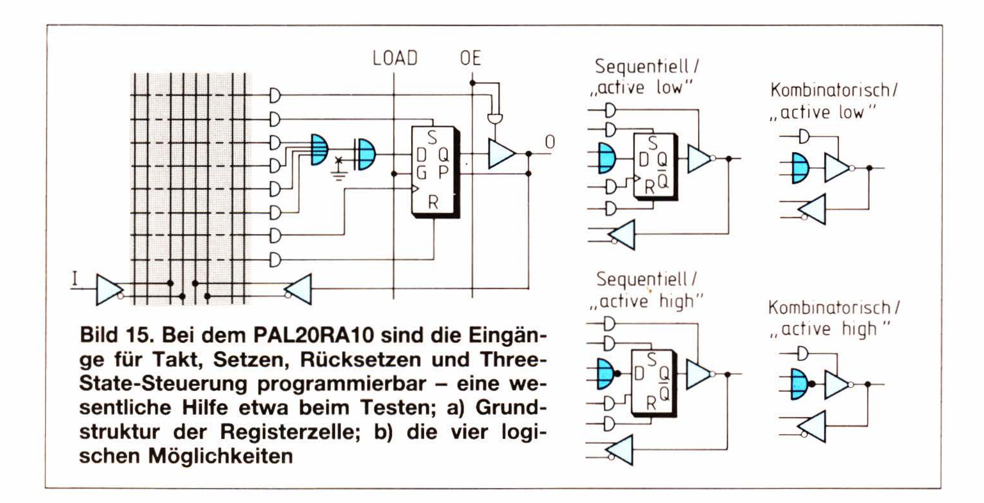


Bild 15. Bei dem PAL20RA10 sind die Eingänge für Takt, Setzen, Rücksetzen und Three-State-Steuerung programmierbar – eine wesentliche Hilfe etwa beim Testen; a) Grundstruktur der Registerzelle; b) die vier logischen Möglichkeiten

rungen im programmierbaren ODER-Array des jeweiligen Ausgangspaares programmiert und, z. B. während der Entwicklung, ein bereits programmierter oder nicht mehr benötigter Produkt-Term vom Gesamt-Array abgetrennt und an dessen Stelle ein anderer verwendet.

Mit dem Abtrennen von Produkt-Termen wird auch der Stromverbrauch des gesamten Bauelementes verringert. Nachdem die Logik einmal fertig erstellt ist, können dann sämtliche Produkt-Terme, die im Logikpattern nicht verwendet sind, vom Stromverbrauch abgekoppelt werden.

PALs mit asynchronen Register-Funktionen

Bei den Register-PALs der herkömmlichen Standard-Familien schaltete ein Takteingang synchron sämtliche im Baustein vorhandenen Flipflops. Diese D-Register ohne Setz- und Rücksetzeingänge konnten deshalb auch nur synchron initialisiert werden. Auf vielfachen Kundenwunsch entwickelte Monolithic Memories nun ein asynchrones Register-PAL, das jetzt als Typ PAL20RA10 verfügbar ist. Bei diesem Baustein finden wir 8 PTs pro Ausgang, die aber anders als bei den bisherigen PAL-Typen verwendet sind.

Vier Produkt-Terme führen über ein Exklusiv-ODER-Gatter zur Programmierung der Ausgangspolarität in den D-Eingang des Flipflops. Die restlichen PTs sind für programmierbare Takt-, Setz-, Rücksetz- und Freigabe-Funktionen verwendbar (*Bild 15*).

Die Rückkopplung der Ausgänge erfolgt bei diesem PAL-Typ von den Pins, das heißt, daß auch die Registerausgänge als Eingänge verwendet werden können.

Werden die Setz- und Rücksetz-Eingänge gleichzeitig aktiv geschaltet, so wird das Register transparent, und man erhält einen kombinatorischen Ausgang. Das Register läßt sich aber jederzeit wieder aktivieren, wenn einer der beiden Steuereingänge des Flipflops (Set/Reset) inaktiv wird.

Um eine sequentielle Schaltung mit der Komplexität eines asynchronen PALs nach der Programmierung mit einem individuellen Logikpattern funktionell testen zu können, ist es möglich, über eine spezielle „PRELOAD"-Einrichtung jedes Register mit einem beliebigen Logikpegel vorzuladen. Damit kann z. B. eine Schaltung auf wichtige Übergangsfunktionen geprüft werden, um sicherzustellen, daß eine sequentielle Schaltung nach einer unerlaubten Sprungfunktion auch wieder in die richtige Sequenz zurückfindet.

Nachdem die Ausgänge mit dem „ENABLE"-Eingang in den Three-State-Zustand gebracht sind, werden mit einem „Low-Pegel" am „PRELOAD"-Eingang die Daten, die am Ausgang der Register anliegen, in die Flipflops übernommen. Außer für Prüfzwecke läßt sich diese Eigenschaft auch noch für logische Schaltungen nutzen, um z. B. Zustände der Außenwelt (Bus-System) über ein Vorladen in den Registern zwischenzuspeichern bzw. zu beurteilen.

PALs mit „Register-Bypass"-Schaltung

Die logischen Zellen einiger PALs können das Signal über das D-Register oder kombinatorisch an den Ausgang führen. Die Entscheidung trifft ein 2:1-Multiplexer, dessen Steuerung über eine weitere Sicherung erfolgt. Ist diese Sicherung intakt, laufen die Signale vom kombinatorischen Array in den D-Eingang des Registers. Ist sie dagegen programmiert, und wird das Signal über den Multiplexer am Register vorbeigeführt, kann das Flipflop trotzdem als „Buried Register" weiter verwendet werden. Diese Funktion ist besonders für Steuerungsaufgaben außerordentlich wichtig (*Bild 16*).

Bild 16. Die von einem programmierbaren Multiplexer (2:1) steuerbare Funktion „Register Bypass" erlaubt ein Umgehen der Speicher. Daraus ergeben sich u. a. kürzere Laufzeiten und ein geringerer Stromverbrauch, wenn die Register in der jeweiligen Anwendung nicht benötigt werden

Dipl.-Ing. Rainer Hövelmann
Dipl.-Ing. Ingo Knacke

IFL-Bausteine in der Praxis

Diskrete Schaltungen mit Gattern, Puffern und Flipflops kann man häufig durch eine integrierte Semi-Kundenschaltung ersetzen. Geeignete Bausteine werden unter den verschiedensten Bezeichnungen angeboten. Als Überbegriff verwendet man oft das Kürzel IFL. Es steht für „Integrated Fuse Logic". Wie man ein konkretes Problem mit IFL-Bausteinen angeht, zeigt dieser Beitrag.

IFLs sind Standard-Bauelemente, die in großen Stückzahlen unprogrammiert hergestellt werden. Durch Trennen von Schmelzpfaden, die im Urzustand intakt sind, wird die vom Anwender gewünschte Funktion programmiert. Das geschieht durch Standardprogrammiergeräte, die durch definierte Ströme und Spannungen die nicht erforderlichen Verbindungen trennen.

Voraussetzung für die Programmierung ist die Umsetzung des elektrischen Schaltplanes in
- Boole'sche Gleichungen oder
- eine Auflistung der Eingangs- und Ausgangsbedingungen (sie werden sinnvollerweise tabellarisch angelegt).

Die Eingabe in das Programmiergerät geschieht entweder manuell oder aus einem Rechnersystem. Auf diese Weise ist es möglich, direkt am Laborplatz einige wenige Bausteine oder in der Fertigung große Mengen in kurzer Zeit zu programmieren.

Für IFLs werden verschiedene Software-Pakete angeboten, die das Entwickeln insbesondere komplexer Schaltungen erleichtern. Diese Hilfsmittel laufen auf gängigen Rechnersystemen und auf Personal-Computern.

Ausgangspunkt ist auch hier, daß der Schaltungsentwurf in Form Boole'scher Gleichungen oder einer Zustandsbeschreibung vorliegt. Nach der Definition der Anschlüsse und der Ein-/Ausgänge folgt z. B. die Eingabe der Gleichungen. Die Software übernimmt dann folgende Aufgaben:
- Syntax-Prüfung auf fehlerhafte Eingabe,
- Minimierung der Produkt-Terme,
- Fehlermeldung-/-kennzeichnung,
- Dokumentation aller Vorgänge,
- Erzeugung der Programmiertabelle,
- Erzeugung einer Standard-Datei mit Programmierdaten zur Weiterverarbeitung.

Der Rechner programmiert danach den Baustein ohne weiteren Eingriff.

Die Software-Pakete enthalten außerdem Simulatoren, die eine Überprüfung der Funktionen gestatten. Auch ein manueller Test ist möglich. Es werden dann bei Eingabe der Eingangsvariablen die zugehörigen Ausgangsgrößen errechnet.

1 Grundlegendes

Der Übersichtlichkeit halber wurde in den Schaltbildern eine Form gewählt, mit der die Vielzahl von Verbindungen einfach und klar dargestellt werden kann (*Bild 1*).

IFLs unterscheiden sich prinzipiell von den bekannten bipolaren PROMs oder den MOS-EPROMs dadurch, daß sowohl der Eingangs- als auch der Ausgangsbereich

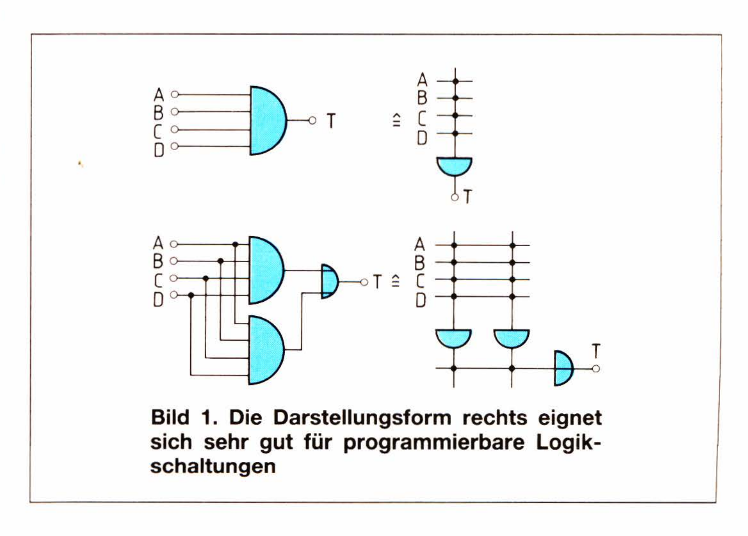


Bild 1. Die Darstellungsform rechts eignet sich sehr gut für programmierbare Logikschaltungen

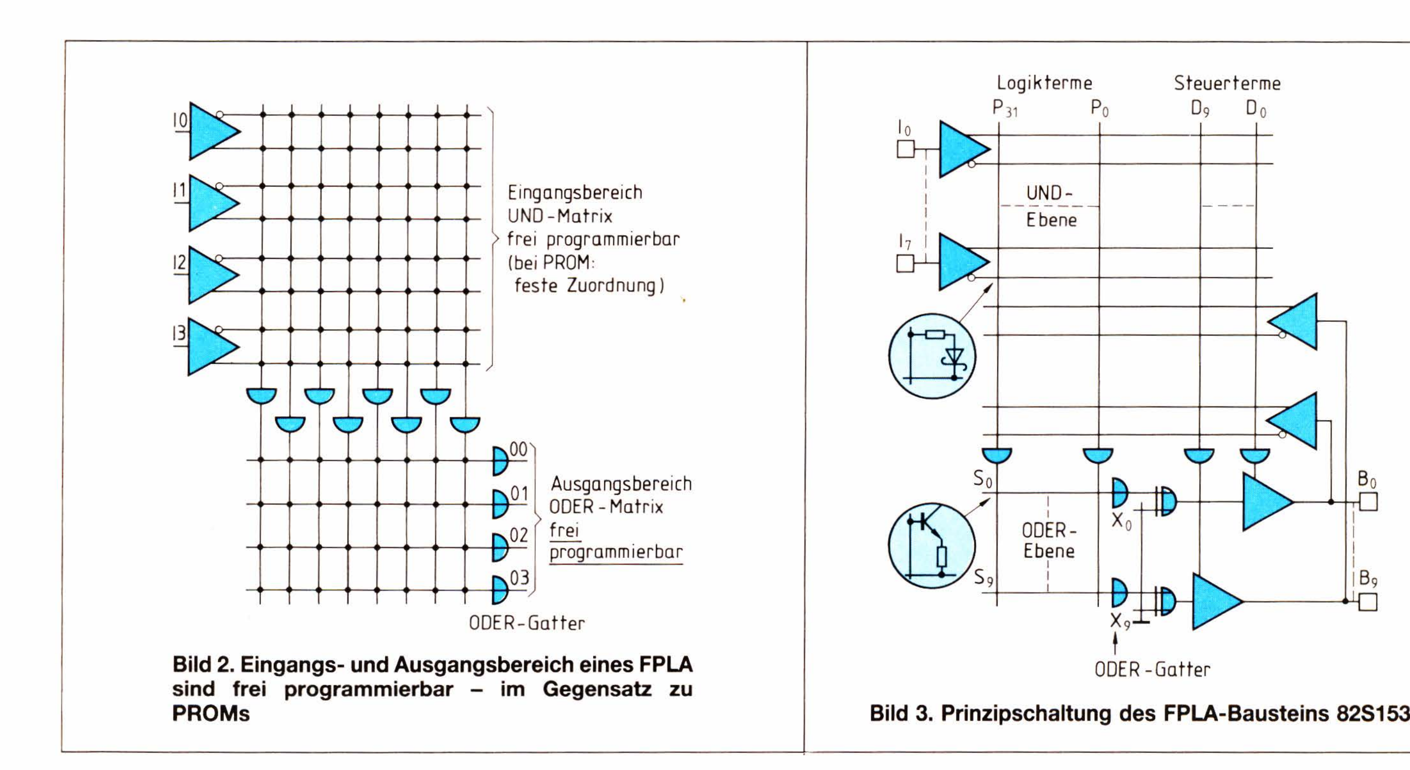


Bild 2. Eingangs- und Ausgangsbereich eines FPLA sind frei programmierbar – im Gegensatz zu PROMs

Bild 3. Prinzipschaltung des FPLA-Bausteins 82S153

programmierbar ist (*Bild 2*). Hierdurch wird eine wesentlich höhere Flexibilität erreicht, die für die Realisierung einer Vielzahl von Funktionen notwendig ist.

Die IFL-Reihe von Valvo z. B. umfaßt drei Familien, die sich im wesentlichen in der Anzahl ihrer Anschlüsse voneinander unterscheiden (20, 24, 28 Anschl.).

Innerhalb dieser Reihen gibt es Typen unterschiedlicher Komplexität:

- FPGA: frei programmierbares Gate-Array
- FPLA: frei programmierbares Logik-Array
- FPLS: frei programmierbares Logik-Sequencer

Abhängig von ihrer Komplexität haben die IFLs verschiedene Bereiche, die getrennt voneinander programmiert werden können:

- Wahr- und Komplement-Eingänge,
- UND-Gatter-Ebene,
- ODER-Gatter-Ebene,
- Ex-ODER-Gatter,
- Komplement-Array,
- getaktete S/R-, J/K-, -D und T-Flipflops.

Diese Bereiche können zu „Einebenen"(UND)- bzw. „Zweiebenen"(UND/ODER)-Logiken oder gar sequentiellen Schaltungen verknüpft werden.

Betrachten wir einmal das 20polige FPLA 825153 (*Bild 3*). Es ermöglicht die Realisierung von „Zweiebenen"-Logik in der Firm:

$$B_0 = (I_0 * I_1 * \ldots) + (I_1 * I_3 * \ldots) + \ldots$$

Die acht Eingänge $I_0 \ldots I_7$ werden über Puffer – Wahr oder im Komplement programmiert – in die UND-Ebene geführt. Diese besteht aus sich kreuzenden Leiterbahnen, deren Kreuzungspunkte im unprogrammierten Zustand miteinander verbunden sind. Getrennt werden nur solche Verbindungen, die für die Funktion nicht benötigt werden. *Bild 4* veranschaulicht das.

Die 32 Produkt-Terme ($P_0 \ldots P_{31}$) werden über UND-Gatter in die zweite Logikebene, die ODER-Ebene, geführt. Dort können bis zu 32 Terme auf jedes der ODER-Gatter programmiert werden. Ein mehrfaches Ausnutzen eines Termes ist möglich.

Die programmierbaren Ex-ODER-Gatter an den Ausgängen gestatten es, die einzelnen Ausgänge als „active high" oder „active low" zu definieren.

Zusätzliche Flexibilität bieten die Schaltungen durch ihre variablen programmierbaren Ein-/Ausgänge ($B_0 \ldots B_9$). Über die Steuerterme wird bestimmt, welcher Ausgang gesperrt (also Eingang) wird. Dies muß nicht statisch, sondern kann durchaus auch dynamisch in Abhängigkeit von den Eingangsvariablen über die Steuerterme erfolgen.

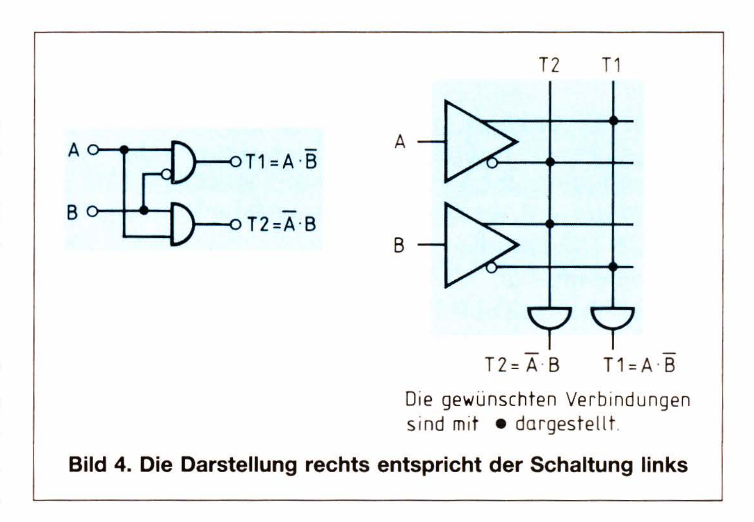


Bild 4. Die Darstellung rechts entspricht der Schaltung links

2 Anwendungsbeispiel

Die Aufgabe soll darin bestehen, auf einer 7-Segment-Anzeigeeinheit die für IFL-Wahrheitstabellen notwendigen Zeichen O, L, H, –, A und · darzustellen. Eingangssignal ist ein 4-Bit-Code. *Bild* 5 zeigt die Zuordnung der Codes zu den Zeichen.

Die gewählte Anzeige hat eine gemeinsame Katode, d. h. ein H in der Tabelle bedeutet, daß das Segment leuchtet. Die Boole'schen Gleichungen 1...6 beschreiben die Aufgabe. *Bild 6* zeigt die zugehörige Schaltung.

$$T = \overline{A} * \overline{B} * \overline{C} * \overline{D} + A * \overline{B} * C * D \quad (1)$$
$$U = V = \overline{A} * \overline{B} * \overline{C} * \overline{D} + \overline{A} * B * \overline{C} * D + A * \overline{B} * C * D \quad (2)$$
$$W = \overline{A} * \overline{B} * \overline{C} * \overline{D} + A * B * \overline{C} * D \quad (3)$$
$$X = Y = \overline{A} * \overline{B} * \overline{C} * \overline{D} + \overline{A} * B * \overline{C} * D + A * B * \overline{C} * D + A * \overline{B} * C * D \quad (4)$$
$$Z = \overline{A} * \overline{B} * C * D + A * \overline{B} * C * D + \overline{A} * B * \overline{C} * D \quad (5)$$
$$P = \overline{A} * B * C * D \quad (6)$$

Nun muß man sich für einen IFL-Typ entscheiden. Da die Eingangssignale permanent verfügbar sind, ist eine Speicherung des Ausgangssignals, d. h. Anwendung eines FPLS, nicht nötig. Wegen der ODER-Verknüpfungen in den Ausgangssignalen und der sich wiederholenden Terme in den einzelnen Gleichungen ist ein FPLA die geeignete Lösung.

An den Gleichungen erkennt man, daß sicher nicht mehr als 32 Produkt-Terme für die Implementierung nötig sind. Der kleinste und preisgünstigste Typ ist also ausreichend. Damit fällt die Entscheidung zugunsten des Typs 82S153.

Als nächstes legt man die Ein- und Ausgänge fest. Folgende Zuordnung soll gelten:

A...D:	Eingänge I0...I3,	Anschl. 1...4
T...Z, P:	Ports B0...B7,	Anschl. 9, 11...17

Die bidirektionalen Ports B0...B7 müssen zu diesem Zweck als feste Ausgänge geschaltet werden.

Die Übertragung der logischen Gleichungen in die Programmiertabelle geschieht nun in einfacher Weise dadurch, daß in der UND-Matrix entsprechend den Eingangssignalen H, L, – (*don't care*) oder 0 (*inactive*) eingesetzt wird. In der ODER-Matrix wird jeweils ein A gesetzt, wenn ein Produkt-Term auf einen Ausgang gelangen soll (d. h. die Verbindung bleibt intakt), andernfalls wird ein Punkt gesetzt. Für Gl. (6) heißt das z. B.:

I0	= L
I1 = I2 = I3	= H
I4...I7	= –
B(0)7	= A
B(0)0...B(0)6	= ·

Für die Steuerung der bidirektionalen Ports sind zusätzliche Steuerterme vorhanden. Ist ein Steuerterm wahr, wird der zugehörigen Ausgangspuffer aktiviert, d. h. das Eingangssignal des Puffers wird auf den Ausgang durchgeschaltet.

Ist der Steuerterm nicht wahr, ist der Puffer hochohmig (*Tri-State*), und der Anschluß kann als Eingang benutzt werden.

Zeichen	Code D C B A	Segmente T	U	V	W	X	Y	Z	P
0	L L L L	H	H	H	H	H	H	L	L
H	H L H L	L	H	H	L	H	H	H	L
L	H L H H	L	L	L	H	H	H	L	L
–	H H L L	L	L	L	L	L	L	H	L
A	H H L H	H	H	H	L	H	H	H	L
•	H H H L	L	L	L	L	L	L	L	H
	I3 ... I0	B0	B1	B2	B3	B4	B5	B6	B7

(Segmentanordnung: T oben, U rechts oben, V rechts unten, W unten, X links unten, Y links oben, Z Mitte, P Punkt)

Bild 5. So muß das Eingangssignal codiert werden

Damit ist es möglich, die bidirektionalen Ports sowohl als feste Ein- oder Ausgänge zu benutzen als auch über eine logische Bedingung zwischen Ein- und Ausgang umzuschalten.

Um B0...B7 zu festen Ausgängen zu machen, müssen wir in den Steuertermen D0...D7 alle Eingänge „–" setzen. Damit wird intern die D-Leitung H, der Steuerterm ist immer wahr (für einen festen Eingang sind alle Eingangssignale 0 zu setzen, so daß der Term nie wahr wird).

Die Polarität des Ausgangssignals ist bei IFL-Schaltungen programmierbar. Programmiert man die Ausgangspolarität auf „active high", bringt ein A in der Tabelle den Ausgang in den H-Zustand. Ist jedoch die Ausgangspolarität auf „active low" gesetzt, verursacht dasselbe A ein L am Ausgang.

Wir sind hier von einer Anzeige mit gemeinsamer Katode ausgegangen. Wenn wir also die Ausgangspolarität für B0...B7 auf H setzen, haben wir damit die Darstellung gemäß Bild 5 zurückgewonnen.

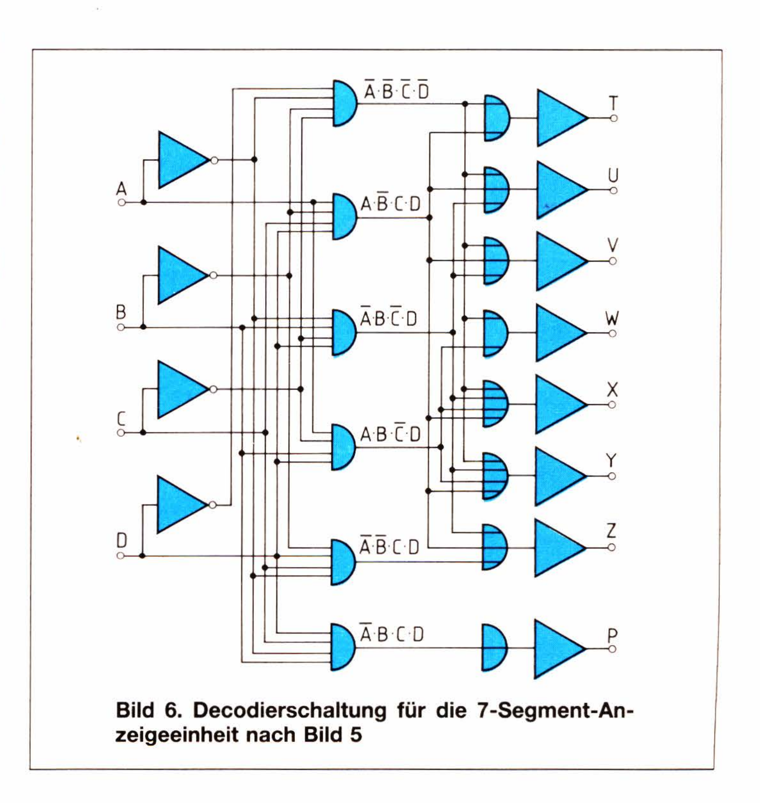
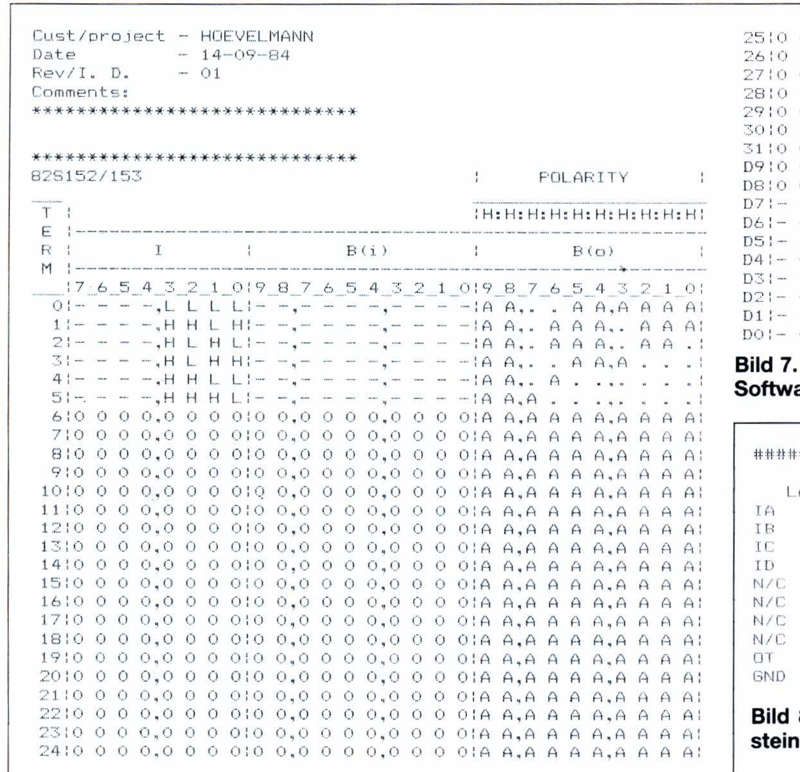


Bild 6. Decodierschaltung für die 7-Segment-Anzeigeeinheit nach Bild 5



```
Cust/project  - HOEVELMANN
Date          - 14-09-84
Rev/I. D.     - 01
Comments:
******************************

******************************
82S152/153                                    !      POLARITY      !
                                              !H:H:H:H:H:H:H:H:H:H!
 T !
 E !--------------------------------------------------------------
 R !        I          !        B(i)          !       B(o)        !
 M !--------------------------------------------------------------
   !7 6 5 4 3 2 1 0!9 8 7 6 5 4 3 2 1 0!9 8 7 6 5 4 3 2 1 0!
  0!- - - -,L L L L!- -,- - - -,- - - -!A A,. . A A,A A A A!
  1!- - - -,H H L H!- -,- - - -,- - - -!A A,. A A A,. A A A!
  2!- - - -,H L H L!- -,- - - -,- - - -!A A,. A A A,. A A .!
  3!- - - -,H L H H!- -,- - - -,- - - -!A A,. . A A,A . . .!
  4!- - - -,H H L L!- -,- - - -,- - - -!A A,. A . .,. . . .!
  5!- - - -,H H H L!- -,- - - -,- - - -!A A,A . . .,. . . .!
  6!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
  7!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
  8!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
  9!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 10!0 0 0 0,0 0 0 0!Q 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 11!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 12!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 13!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 14!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 15!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 16!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 17!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 18!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 19!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 20!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 21!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 22!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 23!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 24!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 25!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 26!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 27!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 28!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 29!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 30!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A'A,A A A A,A A A A!
 31!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!A A,A A A A,A A A A!
 D9!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!
 D8!0 0 0 0,0 0 0 0!0 0,0 0 0 0,0 0 0 0!
 D7!- - - -,- - - -!- -,- - - -,- - - -!
 D6!- - - -,- - - -!- -,- - - -,- - - -!
 D5!- - - -,- - - -!- -,- - - -,- - - -!
 D4!- - - -,- - - -!- -,- - - -,- - - -!
 D3!- - - -,- - - -!- -,- - - -,- - - -!
 D2!- - - -,- - - -!- -,- - - -,- - - -!
 D1!- - - -,- - - -!- -,- - - -,- - - -!
 D0!- - - -,- - - -!- -,- - - -,- - - -!
```

Bild 7. Wahrheitstabelle für den 7-Segment-Decodierer, von der Software erstellt

```
######################## P I N   L I S T ########################

   LABEL      ** FNC **PIN ----------- PIN** FNC **     LABEL
IA            ** I   **  1-!         !-20 ** +5V **VCC
IB            ** I   **  2-!         !-19 ** B   **N/C
IC            ** I   **  3-!    8    !-18 ** B   **N/C
ID            ** I   **  4-!    2    !-17 ** 0   **0P
N/C           ** I   **  5-!    S    !-16 ** 0   **0Z
N/C           ** I   **  6-!    1    !-15 ** 0   **0Y
N/C           ** I   **  7-!    5    !-14 ** 0   **0X
N/C           ** I   **  8-!    3    !-13 ** 0   **0W
0T            ** 0   **  9-!         !-12 ** 0   **0V
GND           ** 0V  ** 10-!         !-11 ** 0   **0U
                           -----------
```

Bild 8. Hier hat der Rechner die Anschlußbelegung des Bausteins ausgedruckt

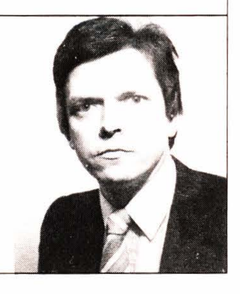
Wie man sieht, wäre es auch einfach möglich, die Anzeige mit gemeinsamer Katode gegen eine mit gemeinsamer Anode auszutauschen. Dazu bräuchte, ohne irgendeine Änderung an den logischen Zusammenhängen, lediglich die Ausgangspolarität geändert zu werden.

Bei einer 7-Segment-Anzeigeeinheit ist es von der Funktion her nicht sinnvoll, eine unterschiedliche Ausgangspolarität für die Steuersignale der einzelnen Segmente zu wählen. In anderen Anwendungsfällen bringt die programmierbare Ausgangspolarität nicht nur mehr Flexibilität in der Realisierung der logischen Funktion, sondern kann auch zu Einsparung von Produkt-Termen führen. Dies sei am Beispiel der Gl. 4 erläutert.



Dipl.-Ing. Ingo Knacke hat als gebürtiger Hamburger in Hamburg Elektrotechnik studiert. Nach dem Abschluß des Studiums ging er 1966 zur Firma Valvo, wo er zunächst im Applikationslabor beschäftigt war. Danach übernahm er Aufgaben im Produkt-Marketing. Zur Zeit leitet er den Bereich „Bipolare Speicher und Logik".

Dipl.-Ing. Rainer Hövelmann wurde in Schwerte/Ruhr geboren. Er studierte Physik in Münster und Elektrotechnik an der Universität Bochum. 1974 trat er in das Applikationslaboratorium der Valvo GmbH ein und beschäftigte sich dort mit Entwicklung und Applikation professioneller integrierter Schaltungen. Seit 1981 ist er in der Valvo-Hauptniederlassung für die Applikation professioneller ICs zuständig.



Sie besteht aus der ODER-Verknüpfung von vier Produkt-Termen entsprechend den vier H in Bild 5. Da X und Y nur zwei L enthalten, wäre es einfacher, die Funktion über die L-Terme darzustellen. Gleichung 4 geht dann über in die Gleichung 4a:

$$\overline{X} = \overline{Y} = \overline{A} * \overline{B} * C * D + \overline{A} * B * C * D \qquad (4a)$$

Die Ausgangspolarität für die Segmente X und Y muß dann auf „active low" gesetzt werden. Das bedeutet, daß X und Y zu L werden, wenn die Gl. 4a wahr ist; in allen anderen Fällen sind die Ausgänge H. Das entspricht der gleichen Funktion wie der vorherigen nach Gl. 4. Der Vorteil ist, daß hier nur zwei Produkt-Terme statt vorher vier benötigt werden. Wenn die zwei zusätzlichen Produkt-Terme nicht wie hier im Beispiel sowieso vorhanden wären, sondern extra erzeugt werden müßten, würde die Ausnutzung der programmierbaren Ausgangspolarität also zur Einsparung von zwei Produkt-Termen führen.

Nun sind alle notwendigen Festlegungen getroffen; der zu definierende Decodierer ist vollständig beschrieben. Die erstellte Wahrheitstabelle (*Bild 7*) kann nun in ein Programmiergerät eingegeben und zur Erzeugung eines Musters benutzt werden.

Selbstverständlich läßt sich der geschilderte Ablauf auch über die Software abwickeln. Hierzu wären lediglich die Booleschen Gleichungen einzugeben. *Bild 8* zeigt die Anschlußbelegung als Rechnerausdruck.

Die zu realisierende Funktion hat lediglich 6 der 32 Produkt-Terme erfordert. Der freibleibende Rest kann für andere Funktionen ausgenutzt werden, ebenso wie die nicht benutzten Eingänge und bidirektionalen Ports.

Willibald Voldan

Definition programmierbarer Logik mit Boole'schen Gleichungen

Eine der vielen Realisierungsmöglichkeiten für digitale Netzwerke wäre es, die gesamte Wahrheitstabelle der Schaltung in einem dementsprechenden logischen Array abzulegen. Digitale Speicher wie z. B. PROMs, RAMs usw. eignen sich für dieses Verfahren, da sie, bedingt durch ihre Struktur, sämtliche Eingangskombinationen zu jedem ihrer Ausgänge decodieren. Dabei ordnet man jeder Eingangsvariablen eine Stelle des Adreßwortes zu und jeder Ausgangsfunktion eine Stelle des Speicherwortes. Ein 512×4-Bit-PROM verfügt also über neun (2^9 = 512) Eingangsvariable und vier Ausgänge. Ein Speicherbaustein, egal welcher Technologie und Konfiguration, stellt für den gegebenen Adreßraum immer einen kompletten Wortraum zur Verfügung, ein PROM kann also zu jedem seiner Ausgänge jede nur denkbare logische Verknüpfung bilden. Der Nachteil dieses Verfahrens besteht darin, daß bereits für wenige Eingänge große Arrays notwendig sind. Werden z. B. zehn Eingänge für die Logikschaltung benötigt, ist bereits ein PROM mit 1024 × 4 (oder × 8) Bit nötig. Üblicherweise erfordern digitale Schaltungen aber weit mehr als nur zehn Eingänge, wodurch eine Realisierung mit Speichern nicht mehr möglich ist. Deshalb wurden „programmierbare logische Anordnungen" entwickelt, die digitale Netzwerke mit vielen Ein- und Ausgängen realisieren.

Die PAL-(Programmable *Array Logic*)Familie von Monolithic Memories umfaßt bereits heute Bausteine in 20-, 24-, 40- und 84-Pin-Gehäusen mit einem Spektrum von Bauelementen mit 6, 8, 10, 12, 14, 16, 18, 20, 32 und 64 Eingängen.

Geht man von einer Funktion in disjunktiver Form aus, z. B.:

Y = (X1 * /X2) + (X2 * X3 * X5) + (/X4 * X6)

(*...steht für die logische „UND"-, +...für die logische „ODER"-Verknüpfung)

erhält man die allgemeine Struktur dieser Bausteine.

Die einzelnen Produktterme werden durch UND-Gatter gebildet, an die die benötigten Eingänge (invertiert und nicht invertiert) angeschlossen werden. Ein ODER-Gatter verknüpft die Produktterme zu dem gewünschten Summenterm.

Ein PAL hat z. B. 64 UND-Gatter pro Produktterm, die im unprogrammierten Zustand über Sicherungen mit allen ODER-Gattern eines Ausganges verbunden sind. Die Anzahl der Produktterme in den PAL-Bausteinen ist für verschiedene Typen unterschiedlich.

Braucht man für eine Ausgangsfunktion nur zwei Eingänge, so muß man die Verbindung der Eingänge zu den restlichen 62 UND-Gattern abtrennen.

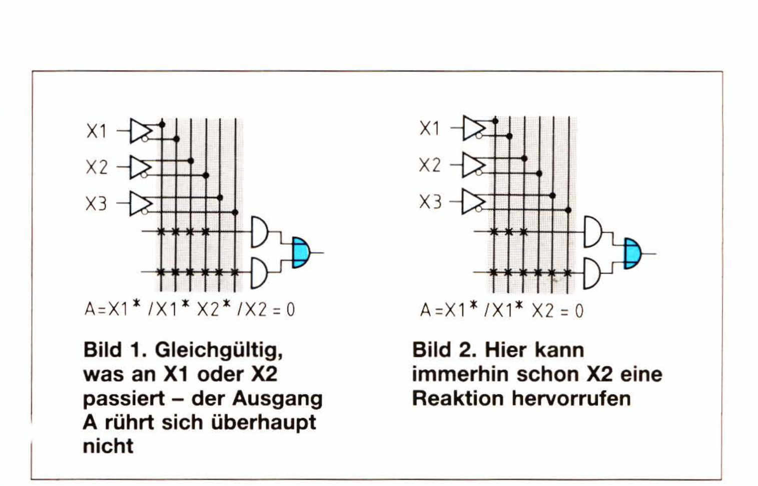


Bild 1. Gleichgültig, was an X1 oder X2 passiert – der Ausgang A rührt sich überhaupt nicht

Bild 2. Hier kann immerhin schon X2 eine Reaktion hervorrufen

In *Bild 1* ist eine Anordnung gezeigt, die durch folgende Gleichung beschrieben werden kann:

A = X1 * /X1 * X2 * /X2 = 0

Hier sind alle Sicherungen in dieser Zeile intakt, also im unprogrammierten Zustand. Trennt man die Sicherung für /X2, ergibt sich A = X1 * /X1 * X2 = 0 wegen X1 * /X1 = 0 (*Bild 2*).

Solange beide Spaltenleitungen von wenigstens einem Eingang an einer Zeilenleitung (Produktterm) lie-

gen, ergibt sich für diese UND-Verknüpfung und auch das daran angeschlossene ODER-Gatter eine 0.

Eine feste 0 an einem ODER-Gatter-Eingang bedeutet aber „Don't Care" und ist damit nicht wirksam. Dieser Umstand ist für die Definition der PAL-Bauelemente besonders wichtig.

Trennt man im obigen Beispiel auch noch die Sicherung für X1, ergibt sich nun folgende Gleichung, die einem logisch sinnvollen Produktterm entspricht (*Bild 3*):

A = /X1 * X2.

Wird zusätzlich noch X2 programmiert, bleibt A = /X1 und X1 hat keinen Einfluß mehr auf das UND-Array (*Bild 4*).

A = /X1 * X2

Bild 3. Auch X1 hat jetzt eine Wirkung auf den Ausgang

A = /X1

Bild 4. Ein simpler Inverter für das Signal an X1

A = X1 * /X2 + /X1 * X2

Bild 5. Das Exklusiv-ODER-Gatter. Nur ein Eingang darf aktiv sein, damit A=1 wird

In *Bild 5* ist die PAL-Schaltung für eine Exklusiv-ODER-Verknüpfung mit der Gleichung

A = X1 * /X2 + /X1 * X2

gezeigt.

Jede Gleichung der Boole'schen Algebra in einer Minterm-Form kann in einem PAL-Baustein realisiert werden, solange die Anzahl der Produktterme, die im Baustein vorhanden sind, ausreicht, um diese logischen Funktionen unterzubringen. Viele Methoden der Logikerstellung, wie z. B. das Karnaugh-Diagramm oder McCluskey, führen zu einer Minterm-Anordnung.

Für PAL-Bausteine mit „Active-High"-Ausgängen werden Terme mit „1" zusammengefaßt und minimiert, bei PALs mit „Active-Low"-Ausgängen dementsprechend auf „0". Bei PAL-Bausteinen mit programmierbarer Ausgangspolarität kann die Ausgangskonfiguration gewählt werden, die weniger Produktterme benötigt.

Mit PAL-Bauelementen wird der Ausdruck „programmierbare Logik" am häufigsten verwendet. Dieser Ausdruck beschreibt eine UND-ODER-Anordnung, wobei die UND-Funktion so viele Eingänge besitzt, wie der gesamte PAL-Baustein aufweist. Ein wesentliches Merkmal programmierbarer Logik ist die „allumfassende Verknüpfbarkeit" der Eingänge, d. h., jeder Eingang kann beliebig oft zu jedem der vorhandenen Ausgänge verbunden werden. Die Bezeichnung Produkt-Term wird mit PT abgekürzt.

Am Anfang der programmierbaren Logik wurden diese Bauelemente manuell über eine Sicherungsmatrix erstellt und programmiert. Die zweidimensionale Matrix eines PALs kann in eine binäre oder hexadezimale Wertetabelle übertragen werden. Diese Wertetabelle, in ein Programmiergerät eingegeben, ermöglicht das Programmieren des Bausteins mit der entsprechenden Logik. Diese Methode ist offensichtlich besonders zeitaufwendig und fehlerbehaftet.

Digitale Systeme sind aber eindeutig über Boole'sche Zustandsgleichungen definierbar. Dieses Verfahren erlaubt eine Simulation, ist rechnerunabhängig und sofort auf jedes Computersystem übertragbar. Die mit Hilfe der Boole'schen Algebra formulierten logischen Gleichungen sind Ausgangspunkt für die Programmierung des PAL-Bausteins mittels PALASM.

Die Eingabe des PAL-Typs, der gewählten Anschlußbelegung und der logischen Gleichungen laut Boole'scher Algebra werden entsprechend dem Flußdiagramm in ein für das Programmiergerät geeignetes Datenformat übersetzt und können sofort in eine Programmiermaschine überspielt werden.

Im *JEDEC-Format*, dem Datenformat für programmierbare Logik, kann die Sicherungsmatrix von beliebig komplexen Bausteinen mit allen nur denkbaren programmierbaren Eigenschaften einfach beschrieben werden. Noch dazu ermöglicht es dieses Format, beliebigen Text als Kommentar und auch die Testvektoren für die Funktion des Bauelementes einzubinden. Das JEDEC-Format entstammt einer gemeinsamen Entwicklung aller Hersteller programmierbarer Logikbausteine zusammen mit den Programmer-Produzenten (s. a. Beitrag „Das JEDEC-Format in Theorie und Praxis").

Logikerstellung mit Hilfe Boole'scher Gleichungen

Zur Entwicklung einer Assembliersprache für programmierbare Logik müssen vorher die Operanden des PAL-Assemblers festgelegt werden:

=	kombinatorisches Gleichheitszeichen
:=	sequentielles Gleichheitszeichen
/	logisches Komplement (Negierung)
*	logische UND-Funktion (Produkt)
+	logische ODER-Funktion (Summe)
:+:	Exklusiv-ODER-Funktion
:*:	Exklusiv-NICHT-ODER-Funktion
IF( )	Three-State-Bedingung
;	Kommentarzeichen

Mit Hilfe dieser einfachen Operanden können nun sämtliche Grundfunktionen der Boole'schen Algebra definiert werden.

Beispiel:

PAL12P6	PAL Type
C D F G M N P Q I GND J K L R S H E B A UCC	PAL-Anschluß- belegung
EQUATIONS	logische Gleichungen
B = /A	;INVERTER
E = C*D	;UND-FUNKTION
H = F+G	;ODER-FUNKTION
L = /J+/K	;NAND-FUNKTION
S = /M*/N	;NOR-FUNKTION
R = P*/Q + /P*Q	;EXKLUSIV-ODER-FUNKTION

Aus diesen einfachen Grundfunktionen lassen sich nun beliebig komplexere Funktionen aufbauen. Nachfolgend werden anhand praktischer Beispiele sämtliche Basis-Funktionen der digitalen Schaltungstechnik beschrieben.

Chip-Select/Memory-Decoder

Diese Anwendung ist eine oft verwendete Funktion programmierbarer Logik. Grundsätzlich können alle logischen Netzwerke als Decoder oder Encoder bezeichnet werden. Die Funktion einer solchen Anordnung ist einfach; sie enthält bloß einfache UND-Funktionen. Die Ausgänge einer solchen Schaltung sind bei einer bestimmten Sequenz der Eingänge aktiv.

Grundsätzlich kann ein Decoder mit N Eingängen 2^N decodierte Ausgänge und 2^N UND-Gatter implementieren. Jedes UND-Gatter hat eine verschiedene Kombination von Eingängen, die zu einer bestimmten Zeit immer nur einen Ausgang aktiv schalten.

Eine typische Gleichung könnte folgendermaßen aussehen:

```
/CS = A7 */A6*/A5* A4* A3*/A2* A1* A0
;DECODE ADDRESS HEX 09D
```

PAL-Bausteine, die als Chip-Select- oder Memory-Decoder programmiert sind, können bereits eine Vielzahl von herkömmlichen 74-SSI/MSI-Bauelementen in der Funktion ersetzen. Noch dazu sind programmierbare Bausteine wesentlich flexibler in der Anwendung; Änderungen oder Anpassungen an verschiedene Schaltungsprobleme können ohne „Redesign" der Leiterplatte durchgeführt werden.

Multiplexer

Eine Multiplexer-Schaltung selektiert Daten von verschiedenen Kanälen auf einen Ausgang. Ein einfacher 4-zu-1-Multiplexer besitzt vier Dateneingänge und zwei Steuerleitungen, die bestimmen, welcher von den vier Datenkanälen auf den Ausgang geschaltet wird.

```
/1Y = /B*/A*/1C0      ;SELECT INPUT 1C0
    + /B* A*/1C1      ;SELECT INPUT 1C1
    +  B*/A*/1C2      ;SELECT INPUT 1C2
    +  B* A*/1C3      ;SELECT INPUT 1C3
```

Jedes der UND-Gatter besitzt einen Daten-Eingang und eine Kombination der zwei Steuereingänge. Mit einer von vier möglichen Kombinationen der Steuer-Bits ist immer nur ein UND-Gatter, das der Steuer-Kombination entspricht, freigegeben und erlaubt damit den entsprechenden Daten zum Ausgang zu kommen.

Diese Schaltung kann einfach erweitert werden, indem weitere Dateneingänge sowie eine Zahl entsprechender Steuerleitungen hinzugefügt werden (5 bis 8 Dateneingänge erfordern 3 Steuerleitungen, 9 bis 16 brauchen 4 Steuerleitungen).

Komparator (Exklusiv-ODER)

Die Exklusiv-ODER-(XOR-)Schaltung wird oft als steuerbare Inverter-Funktion oder als digitale Vergleichsschaltung bezeichnet. XOR-Gatter mit mehr als zwei Eingängen werden darüber hinaus häufig für Modulo-2-Arithmetik oder zur Generierung von Parity-Signalen verwendet.

Das Exklusiv-ODER-Gatter mit mehreren Eingängen ist aktiv, wenn eine ungerade Anzahl von Eingängen zum XOR auf logisch „high" sind.

Die Exklusiv-ODER-Funktion (der Vergleich) bildet die grundsätzliche Struktur vieler logischer Anordnungen, z. B. arithmetischer Schaltungen, Komparatoren, Zähler und Schieberegister. Nachfolgend sind die logischen Gleichungen einiger digitaler Komparatoren angeführt:

```
NE = A0*/B0 + /A0* B0          ;A0 .NE. B0
EQ = A0* B0 + /A0*/B0          ;A0 .EQ. B0
```

Halb-Addierer

```
S = /A* B + A*/B               ;Summe von A & B
C =  A* B                      ;Übertrag
```

Voll-Addierer

```
S =  A* B* CIN                 ;Summe von A & B & CIN
  +  A*/B*/CIN
  + /A*/B* CIN
  + /A* B*/CIN
C =  A*B + B*CIN + A*CIN       ;Übertrag
```

Halb-Subtrahierer

```
D = /X* Y + X*/Y               ;Differenz von X-Y
B = /X* Y                      ;Übertrag
```

Voll-Subtrahierer

```
D =  X* Y* BIN                 ;Differenz von X-Y-BIN
  +  X*/Y*/BIN
  + /X*/Y* BIN
  + /X* Y*/BIN
B = /X* Y + /X*BIN + Y*BIN     ;Übertrag
```

Sequentielle Logik

Einige PALs enthalten bereits D-Register. Die logischen Gleichungen für sequentielle Schaltungen unterscheiden sich nur unwesentlich von denen der kombi-

natorischen Funktionen. Anstelle des kombinatorischen Gleichheitszeichens wird eine sequentielle Übergangsfunktion mit einem := dargestellt. Damit werden zwei Zeiteinheiten einer Registerfunktion beschrieben (die „Set-up"- und die „Clock-to-Output"-Zeit).

Um die verschiedenen Flipflop-Schaltungen mit dem im PAL eingebauten D-Register zu realisieren, können folgende logische Gleichungen verwendet werden:

D-Flipflop

```
QD := D*/CLEAR              ;D FLIP-FLOP (TRUE)
    + PRESET                ;PRESET Q
```

RS-Flipflop

```
QRS := S*/CLEAR             ;SET-RESET FLIP-FLOP
                            (TRUE)
     + /R* QRS*/CLEAR       ;(QRS=Q)
     + PRESET               ;PRESET Q
```

JK-Flipflop

```
QJK := J*/QJK*/CLEAR        ;J-K FLIP-FLOP (TRUE)
     + /K* QJK*/CLEAR       ;(JKT=Q)
     + PRESET               ;PRESET Q
```

T-Flipflop

```
QT := T*/QT */CLEAR         ;T FLIP-FLOP (TRUE)
    + /T* QT */CLEAR        ;(QT=Q)
    + PRESET                ;PRESET Q
```

Schieberegister

Schieberegister-Schaltungen werden für Multiplikation, Division oder serielle Datenübertragung in der Kommunikation verwendet. Eine Schieberegister-Funktion kann einfach aus einem Satz Multiplexer mit nachgeschalteten Flipflops zur Speicherung der jeweiligen Zustände konstruiert werden. Eine typische Schieberegister-Schaltung kann Daten laden, nach links und rechts schieben und auch einen „HALTE"-Zustand einnehmen (d. h. keine Veränderung der Datenausgänge). Um alle diese Funktionen anwählen zu können, benötigt man zusätzlich zu den Daten- noch zwei Steuer-Eingänge.

Werden Daten in das Schieberegister geladen, ist der dementsprechende Datenweg freigegeben und die an den Datenanschlüssen anliegenden Informationen werden in die Ausgangsregister übernommen.

Die Ausgänge der Flipflops sind über den Multiplexer in den Dateneingang des jeweilig nächstliegenden Registers rückgekoppelt.

Beim Rotieren der Daten nach rechts taktet ein Bit J in J+1. In ähnlicher Weise werden die Daten nach links verschoben, indem das Bit J+1 in das Bit J versetzt wird.

Das „Halten" der Daten wird einfach durch die Rückkopplung des Registers in sich selbst erzielt, wenn dieser Datenpfad vom Multiplexer angewählt ist.

```
/Q0 :=   /S1*/S0*/Q0     ;HOLD Q0
     +   /S1* S0*/Q1     ;SHIFT RIGHT
   :+:    S1*/S0*/L1     ;SHIFT LEFT
     +    S1* S0*/D0     ;LOAD D0
/Q1 :=   /S1*/S0*/Q1     ;HOLD Q1
     +   /S1* S0*/Q2     ;SHIFT RIGHT
   :+:    S1*/S0*/Q0     ;SHIFT LEFT
     +    S1* S0*/D1     ;LOAD D1
/Q2 :=   /S1*/S0*/Q2     ;HOLD Q2
     +   /S1* S0*/Q3     ;SHIFT RIGHT
   :+:    S1*/S0*/Q1     ;SHIFT LEFT
     +    S1* S0*/D2     ;LOAD D2
```

In einer allgemeinen Form, die der PALASM-II verarbeiten kann, ist es möglich, eine Reihe von strukturierten Gleichungen in einer kompakten Sequenz darzustellen.

```
/Q{I=0...7} :=  /S1*/S0*/Q {I}     ;HOLD Q {I}
             +  /S1* S0*/Q {I+1}   ;SHIFT RIGHT
           :+:   S1*/S0*/Q {I–1}   ;SHIFT LEFT
             +   S1* S0*/D {I}     ;LOAD D {I}
```

Makro-Sequenzen für logische Gleichungen helfen, immer wiederkehrende Logikfunktionen in kompakter und übersichtlicher Form darzustellen. Die kompakte Makro-Sequenz

```
Q{I=0...3} = A{I}*S + B{I}*/S
```

ersetzt die vier Gleichungen eines Multiplexers vom Typ 74153:

```
Y0 = A0*S + B0*/S
Y1 = A1*S + B1*/S
Y2 = A2*S + B2*/S
Y3 = A3*S + B3*/S
```

Folgende Gleichungen beschreiben die bekanntesten Interface-Strukturen:

```
IF(/OE) Y{I=0...7}  = A{I}             ;74S/LS244
IF(/OE) /Y{I=0...7} = A{I}             ;74S/LS240
IF(/OE) Q{I=0...7} := D{I}             74S/LS374
IF(/OE) /Q{I=0...7} := D{I}            ;74S/LS534
        Q{I=0...7} := D{I} + CLEAR     ;74S/LS273
```

Zähler-Funktionen

Zähler werden als Statusmaschinen, digitale Verzögerungs-Schaltungen oder Ereignis-Zähler verwendet. Sie bilden die grundlegendste und am meisten gebräuchliche Funktion in der Digitaltechnik.

Der Schlüssel für die Architektur eines Zählers liegt in einer Komparator-Schaltung (Exklusiv-ODER) mit nachgeschalteten Registern. Der Komparator vergleicht die Registerausgänge, um festzustellen, wann ein bestimmter Ausgang umgeschaltet werden soll. In einem Vor-

wärtszähler wird ein Registerausgang auf logisch „High" umgeschaltet, wenn sämtliche vorangegangenen Bits auf „High" sind.

Der niedrigstwertige Registerausgang ändert seinen Zustand mit jedem Taktimpuls.

Zählerstand	Nächster Zählerstand
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	1010
1010	1011
1011	1100
1100	1101
1101	1110
1110	1111
1111	0000

Die typische Zählerfunktion besteht aus der Zählersequenz (Vorwärts, Rückwärts oder beides), Daten zu „halten", Daten in den Zähler zu laden, und das System vorzusetzen oder zu löschen.

Neben dem erforderlichen „Overhead" an Produkttermen für „Clear, Load, Hold, Set, Preset" usw. erfordert das n-te Bit n Produktterme.

```
/Q0 := Q0                ;COUNT   Q0
/Q1 :=  Q0* Q1           ;COUNT   Q1
     + /Q0*/Q1           ;HOLD    Q1
/Q2 :=  Q0* Q1* Q2       ;COUNT   Q2
     + /Q0*/Q2           ;HOLD    Q2
     + /Q1*/Q2           ;HOLD    Q2
```

Steuersignale können in die Zählersequenz einfach über zusätzliche Produktterme in die Gleichungen eingebaut werden. Das niedrigstwertige Bit, zusätzlich mit einer „Clear"- und einer „Data-Load"-Funktion, wird dann in Gleichungsform folgendermaßen dargestellt:

```
/Q0 := Q0*/LOAD      ;COUNT
     + /D0* LOAD     ;LOAD    D0
     + CLEAR         ;CLEAR   Q0
```

Für die einfache Realisierung von Zählerfunktionen, die grundsätzlich aus Vergleichssequenzen bestehen, besitzen einige PAL-Bausteine (16X4, 16A4, 20X4, 20X8 und 20X10) integrierte Exklusiv-ODER-Gatter. Mit diesen Ex-ODER-Gattern kann nun einfach die „HOLD"-Funktion des Zählers gebildet werden. Damit sind beliebig große Kaskaden mit nur zwei Produkttermen (bzw. drei für das Vorwärts- und Rückwärtszählen) für den Zählvorgang selbst möglich:

```
/Q0 := /SET* LD*/D0                     ;LOAD D0
     + /SET*/LD*/Q0                     ;HOLD Q0
   :+: /SET*/LD* CNT*CIN* UP            ;INCREMENT
     + /SET*/LD* CNT*CIN*/UP            ;DECRE-
                                         MENT
/Q1 := /SET* LD*/D1                     ;LOAD D1
     + /SET*/LD*/Q1                     ;HOLD Q1
   :+: /SET*/LD* CNT*CIN*UP* Q0         ;INCREMENT
     + /SET*/LD* CNT*CIN*/UP*/Q0        ;DECREMENT
/Q2 := /SET* LD*/D2                     ;LOAD D2
     + /SET*/LD*/Q2                     ;HOLD Q2
   :+: /SET*/LD* CNT*CIN* UP*           ;INCREMENT
       Q0* Q1
     + /SET*/LD* CNT*CIN*/UP*/          ;DECREMENT
       Q0*/Q1
```

Die Zählerfunktion für mehrere Ausgänge kann in einer kompakten Form so dargestellt werden:

```
Q{I=0...7} :=   Q{I}                ;HOLD Q{I}
           :+:  Q{I-1}*...*Q{0}     ;COUNT
oder
/Q{I=0...7} :=  /Q{I}               ;HOLD Q{I}
            :+:  Q{I-1}*...*Q{0}    ;COUNT
```



Informationen zu den CMOS-PAL's von CYPRESS

Zellstruktur:

Die PAL's von CYPRESS halten ihre Informationen in Form von Ladungen auf einem Floating Gate Transistor. Durch Injektion von Ladungsträgern wird eine Zelle programmiert, nicht durch physikalisches Zerstören einer Sicherung (wie bei den Bipolartypen). Da der Hersteller die Möglichkeit hat, jede Zelle zu programmieren (und anschließend zu löschen), erhält der Anwender 100% Programmierausbeute.

CMOS-Technologie

Ein 1.2 μ CMOS-Prozeß in Verbindung mit der bewährten EPROM-Technologie bedeutet für den Anwender ein hervorragendes Bauteil sowohl bezgl. Performance, Zuverlässigkeit, Testbarkeit und Programmierbarkeit.

Vorteile der CMOS PAL's gegenüber Bipolartechnik

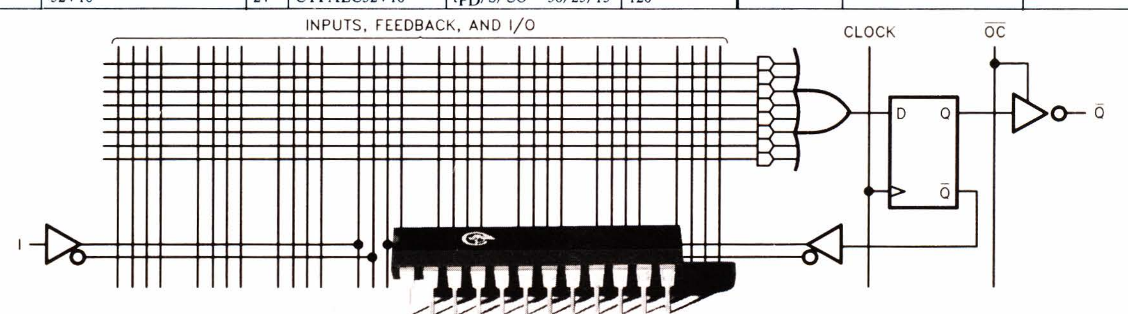
Die meisten Parameter der PAL's von CYPRESS entsprechen denen der Bipolartypen oder übertreffen diese.
Folgende Vorteile sind für den Anwender interessant:

- wesentlich niedrigere Stromaufnahme durch CMOS-Technologie
- geringere Kühlung notwendig
- durch niedrigere Chip-Temperaturen höhere Zuverlässigkeit.

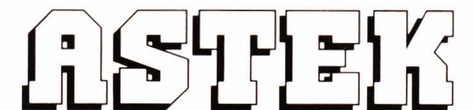
Das CYPRESS-Programm

- ☐ **stat. RAM's** 16 x 4, 256 x 4, 1K x 4, 4K x 4, 4K x 1, 16K x 1
- ☐ **PAL's** 16L8, 16R6, 16R8, 16R4
- ☐ **FiFo's** 64 x 4, 64 x 5
- ☐ **PROM's** 512 x 8, 1K x 8, 2K x 8
- ☐ **Bitslice** 29xx-Familie

	Organization	Pins	Cypress			Closest Competitor		
			Part Number	Speed (ns)	I_{CC} (mA @ ns)	Part Number	Speed (ns)	I_{CC} (mA @ ns)
PALs	16L8	20	CYPALC16L8A	t_{PD} = 25	90			
	16R8	20	CYPALC16R8A	t_S/CO = 20/15	90			
	16R6	20	CYPALC16R6A	t_{PD}/S/CO = 25/20/15	90			
	16R4	20	CYPALC16R4A	t_{PD}/S/CO = 25/20/15	90			
	16L8	20	CYPAL16L8A	t_{PD} = 25	155	PAL16L8A	t_{PD} = 25	180
	16R8	20	CYPAL16R8A	t_S/CO = 20/15	155	PAL16R8A	t_S/CO = 20/15	180
	16R6	20	CYPAL16R6A	t_{PD}/S/CO = 25/20/15	155	PAL16R6A	t_{PD}/S/CO = 25/20/15	180
	16R4	20	CYPAL16R4A	t_{PD}/S/CO = 25/20/15	155	PAL16R4A	t_{PD}/S/CO = 25/20/15	180
	16L8	20	CYPAL16L8A-2	t_{PD} = 35	90	PAL16L8A-2	t_{PD} = 35	90
	16R8	20	CYPAL16R8A-2	t_S/CO = 30/25	90	PAL16R8A-2	t_S/20 = 30/25	90
	16R6	20	CYPAL16R6A-2	t_{PD}/S/CO = 35/30/25	90	PAL16R6A-2	t_{PD}/S/CO = 35/30/25	90
	16R4	20	CYPAL16R4A-2	t_{PD}/S/CO = 35/30/25	90	PAL16R4A-2	t_{PD}/S/CO = 35/30/25	90
	22V10	24	CYPALC22V10	t_{PD}/S/CO = 30/25/15	120	22V10	t_{PD}/S/CO = 25/25/15	180
	32V10	24	CYPALC32V10	t_{PD}/S/CO = 30/25/15	120			



INNOVATIVE TECHNOLOGIEN VON



ELEKTRONIK VERTRIEBS GmbH
Carl-Zeiss-Straße 3 · 2085 Quickborn · Tel. 04106/7 10 84-86 · Tx 21 4082 askd

Ausführliche Unterlagen senden wir Ihnen gern zu. Der „PAL-Application-Brief und das CYPRESS-Datenbuch stehen auf Anfrage zur Verfügung.

Willibald Voldan

LOGASM – Optimierungsprogramm für digitale Schaltungen

Die Voraussetzung für den Entwurf einer digitalen Schaltung ist ein Pflichtenheft, das festlegt, was die entsprechende Anordnung leisten soll. Diese Vorschrift existiert meist in Form einer Zustandstabelle, die die Ausgangssignale bei entsprechenden Eingangskonfigurationen beschreibt. Im allgemeinen ist es nicht schwer, Schaltungen zu finden, die eine gestellte Aufgabe erfüllen können. Das Erstellen einer optimalen Lösung, die mit möglichst wenigen Verknüpfungen auskommt, kann sich jedoch zu einer wahren Sisyphosarbeit auswachsen. Bereits im vorigen Jahrhundert entwickelte G. Boole seine Algebra der Logik, bei der die Variablen nur die beiden Werte 0 oder 1 annehmen können. Seit den Anfängen ihrer Anwendung auf logische Schaltungen wird nach Minimierverfahren gesucht, die eine digitale Problemstellung auf die einfachste Form bringen sollen. Da es sich hierbei um einen rein mechanischen, festen Regeln unterworfenen Vorgang handelt, ist dies eine klassische Aufgabe für den Computer.

Minimierung bei PALs

Eine logische Schaltung mit einer Anzahl von Ein- (N) und Ausgängen (A) läßt sich über eine Verknüpfung von maximal $2^N * A$ beschreiben, falls die Anordnung keine Speicherfunktionen oder Rückkopplungen besitzt.

Innerhalb der Architektur eines beliebigen PROMs realisiert der Adreßdecoder des Bausteins alle Minterme zu jedem seiner Ausgänge. Im Unterschied zum PROM besteht ein PAL aus einem programmierbaren Eingangs-UND-Array, das ein festgelegtes ODER-Ausgangsarray treibt. Diese Architektur ermöglicht zwar beliebig viele Ein- und Ausgänge, hat aber den Nachteil, daß die Anzahl der realisierbaren Produktterme (UND-Gatter) sehr viel kleiner als die Anzahl der möglichen Terme ist, nämlich $PT << 2^N$. So erfordert z. B. der PAL-Baustein 16L8 mit 16 Eingängen zur kompletten Decodierung der Eingänge 2^{16} Produktterme zu jedem seiner Ausgänge. In Wirklichkeit hat das PAL 16L8 insgesamt nur 64 PT, 8 zu jedem Ausgang.

Viele Probleme sind jedoch minimierbar, eventuell unter Einbeziehung von physikalisch nicht möglichen Eingangskombinationen (sogenannte Pseudo-Terme). Dadurch läßt sich die Zahl der benötigten Produktterme reduzieren. Ziel einer Minimierung bei PAL-Bausteinen muß also sein, eine logische Problemstellung mit möglichst wenigen Produkttermen zu realisieren.

In der Vergangenheit wurden vorzugsweise zwei verschiedene Wege zur Minimierung beschritten, die hier kurz beschrieben werden sollen.

Minimierung logischer Gleichungen mit Hilfe algebraischer Operationen

Mit Hilfe der Rechenregeln der Schaltalgebra lassen sich logische Gleichungen umformen und vereinfachen, wobei natürlich die Form mit dem geringsten Aufwand gesucht wird.

Ein logischer Begriff besteht aus einer nichtinvertierten oder invertierten Variablen. Eine logische Gleichung ist aus Variablen zusammengesetzt, die durch logische Operanden getrennt sind. Terme werden oft auch durch Klammerzeichen separiert.

Die Gleichung $F = (a*b) + (c*/a) + c$ besteht aus drei logischen Variablen und drei Termen. Die Konstruktion einer solchen Anordnung ist eine Gleichung im Format „Summe-von-Produkten“, weil die Gleichung aus einer Anzahl von Produkt-Termen zusammengesetzt ist, die in einer ODER-Funktion verbunden sind. Eine Gleichung im Format $F = (a+b) * (a+/b)$ wird als „Produkte-von-Summen“ bezeichnet, weil sie aus einer Reihe von Summentermen (ODER-Funktionen) besteht, die über ein UND-Gatter miteinander verbunden sind.

Eine Boole'sche Gleichung zu vereinfachen, ist der Versuch, die Anzahl der Variablen und Terme zu minimieren. Oft ist aber nur das eine auf Kosten des anderen möglich.

Zur Minimierung können eine ganze Reihe von Theoremen und Gesetzen der Boole'schen Algebra herhalten. Diese Regeln dienen hauptsächlich zur Umformung logischer Gleichungen, bei der oft auch eine Vereinfachung der Logik möglich ist. Zum Beispiel kann die Gleichung

F = (x∗/z) + ((x+y)∗/z)

unter Verwendung dieser Regeln schrittweise minimiert werden:

```
F = (x*/z) + ((x+y) * /z)
  = (/z*x) + (/z*(x+y))
  = /z * (x + (x+y))
  = /z * ((x+x) + y)
  = /z * (x+y)
```

Dieses Verfahren ist zeitaufwendig und daher fehlerbehaftet. Meist kann zwar minimiert werden, das Ergebnis muß aber nicht immer das Optimum darstellen. Wird die Zahl der Eingangsvariablen zu groß, ist diese Methode kaum noch zu gebrauchen.

Das Karnaugh-Diagramm

Für die Darstellung und Berechnung digitaler Schaltfunktionen existieren neben rein algebraischen Verfahren auch grafische Methoden.

Karnaugh-Diagramme basieren ebenfalls auf den Grundlagen der Boole'schen Algebra und stellen nur eine andere Form der Wahrheitstabelle dar. Sie realisieren eine Technik zur Minimierung von logischen Gleichungen im Format „Summe-von-Produkten".

Jeder der möglichen Eingangskombinationen der Wahrheitstabelle wird im Karnaugh-Diagramm eine Adresse zugeordnet, in die dann der jeweilige Ausgangszustand 0 oder 1 eingetragen wird. Benachbarte Felder ändern beim Übergang von einem Feld in das nächstliegende immer nur eine Eingangsvariable. Darauf beruht das Prinzip der Minimierung (*Bild 1*).

Ein Karnaugh-Diagramm kann verschiedene Lösungen hervorbringen, die ebenfalls nicht den minimal möglichen Termen entsprechen müssen. Das Verfahren ist nützlich für max. sechs Variable. Es versagt kläglich, wenn die Zahl der Eingangsvariablen zu groß wird.

Der Logik-Assembler LOGASM

Monolithic Memories hat in Zusammenarbeit mit der Firma EDTZ den Logikassembler „LOGASM" entwickelt, der aus Zustandstabellen sofort die optimale (d. h. minimierte) Lösung in Gleichungsform liefert. Dieses Computerprogramm ist für die Betriebssysteme CP/M (Digital Research) und ISIS-II (Intel) konzipiert und kann auf Standard-µP-Entwicklungssystemen bis zu 15 Eingangsvariable berechnen. Es werden sowohl für kombinatorische wie auch für sequentielle Netzwerke bzw. Schaltwerke Zustandstabellen minimiert und die logischen Gleichungen erstellt, wobei auch für große Zustandstabellen mit vielen Eingangstermen nur kurze Ablaufzeiten notwendig sind.

Die Erstellung der Zustandstabelle (das Eingabeprogramm) erfolgt mit Hilfe des systemeigenen Editors. Die Eingabe der logischen Zustände kann binär, dezimal und auch hexadezimal erfolgen, um die verschiedensten Aufgabenstellungen der digitalen Schaltungsentwicklung zu berücksichtigen. Ein Wechsel des Eingabeformats ist im Programm möglich.

Ein Semikolon am linken Zeilenrand ermöglicht es, Kommentarzeilen vorzusehen, die den Eingabetext übersichtlicher gestalten. Sie werden im Programmablauf völlig ignoriert. Auch Leerzeilen sind erlaubt; sie wirken wie die Kommentare. Ein Punkt am linken Zeilenrand kennzeichnet eine Kommandozeile.

Vor der Eingabe des ersten Terms sind Format und Bitbreite in einer Kommandozeile zu definieren. Diese Kommandozeile besteht aus einem Punkt am linken Zeilenrand, der Formatangabe (BINAER, HEXADEZIMAL oder DEZIMAL) und der Bitbreite. Die Bitbreite ist in dezimaler Schreibweise einzugeben. Während das Eingabeformat im File beliebig oft geändert werden darf, ist die einmal definierte Bitbreite für das ganze Programm gültig.

```
.BINAER 4        Eingabe binär, 4 Bit breite Eingangsterme
.DEZIMAL         Eingabe dezimal
.HEXADEZIMAL     Eingabe hexadezimal
```

Eine große Anzahl möglicher Fehlermeldungen helfen dem Benutzer, Syntaxfehler, die bei der Eingabe passiert sind, schnell zu finden. Dazu kommt die Möglichkeit,

Wahrheitstabelle

A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Karnaugh-Diagramm

A \ BC	00	01	11	10
0	0	1	1	1
1	0	1	0	1

Produkt-Term 1 = B*/C
Produkt-Term 2 = /B*C
Produkt-Term 3 = /A*C
in "Summen-von-Produkten"
F = /A*C+/B*C+B*/C

Von der Wahrheits- oder Zustandstabelle zum Karnaugh-Diagramm. Hier am Beispiel von drei Eingangs- und einer Ausgangsvariablen gezeigt

Makro-Befehle zu definieren und damit die Eingabe wesentlich schneller und übersichtlicher zu gestalten. Mit Hilfe einer Zeile können Hunderte von Eingangstermen definiert werden.

Von der Problemstellung zur Lösung

In diesem Kapitel soll anhand zweier Beispiele gezeigt werden, wie man von einer Problemstellung mit Hilfe des LOGASM über die Zustandstabelle und den Eingabetext zur Problemlösung kommt.

Die Aufgabe

Eine bestehende elektronische Schaltung erzeugt einen 4 Bit breiten BCD-Code (8-4-2-1), der zur Weiterverarbeitung in einen 4 Bit breiten AIKEN-Code umgewandelt werden soll. Es ist also ein Umcodierer von BCD in AIKEN-Code bei je 4 Bit Breite zu entwickeln.

Zustandstabelle

Um das Problem logisch erfassen zu können, erstellt man eine sogenannte Zustands- oder auch Wahrheitstabelle. Auf der linken Seite der Tabelle (Eingangszustände) trägt man alle 4-Bit-Dualzahlen untereinander auf. Um die einzelnen Eingangsterme (bzw. Eingangszustände) besser bezeichnen zu können, schreibt man vor die Dualzahl den ihr entsprechenden Dezimalwert. In der rechten Seite dieser Tabelle notiert man für jeden möglichen Eingangszustand (BCD-Code) den dazu gehörenden 4 Bit breiten Ausgangszustand (AIKEN-Code).

BCD-AIKEN-Codierung

	BCD				AIKEN				
	E4	E3	E2	E1	A4	A3	A2	A1	
0	L	L	L	L	L	L	L	L	
1	L	L	L	H	L	L	L	H	
2	L	L	H	L	L	L	H	L	
3	L	L	H	H	L	L	H	H	
4	L	H	L	L	L	H	L	L	
5	L	H	L	H	H	L	H	H	
6	L	H	H	L	H	H	L	L	
7	L	H	H	H	H	H	L	H	
8	H	L	L	L	H	H	H	L	
9	H	L	L	H	H	H	H	H	
10	H	L	H	L	X	X	X	X	P
11	H	L	H	H	X	X	X	X	S
12	H	H	L	L	X	X	X	X	E
13	H	H	L	H	X	X	X	X	U
14	H	H	H	L	X	X	X	X	D
15	H	H	H	H	X	X	X	X	0

Der BCD-Code (oder auch 8-4-2-1-Code) hat nur die Eingangsterme 0...9. Im AIKEN-Code gibt es auch nur 10 solcher Eingangsterme, die auch Tetraden genannt werden. Die Tetraden 10...15 des BCD-Codes sind nicht belegt und liefern deshalb auch keine zugehörigen Ausgangsterme im Aiken-Code. Man nennt solche unbestimmten Ausgangsterme Pseudotetraden oder Pseudoausgangsterme.

Eingabetext

Jetzt lassen sich mit Hilfe der obigen Tabelle für jedes Ausgangsbit diejenigen Terme angeben, die den gewünschten Ausgangszustand ergeben. Wird auf „Low" minimiert, werden die Terme für log. „0" angegeben, wird auf „High" minimiert, dementsprechend auf log. „1".

In unserem Beispiel soll auf „1" minimiert werden. Daher lauten die Eingangsterme folgendermaßen:

```
Bit 4:  0101,  0110,  0111,  1000,  1001
Bit 3:  0100,  0110,  0111,  1000,  1001
Bit 2:  0010,  0011,  0101,  1000,  1001
Bit 1:  0001,  0011,  0101,  0111,  1001
```

Die Pseudoterme in diesem Beispiel lauten für alle vier Ausgangsbits gleich:

```
Pseudo  1010,  1011,  1100,  1101,  1110,  1111
```

Für die Erstellung logischer Gleichungen zur Decodierung des BCD- in den AIKEN-Code wird nun mit dem systemeigenen Editor der Eingabefile erstellt (Die Ausgänge sollen auf „High-Aktiv" minimiert werden):

```
;  BIT 4        ;BIT3         ;BIT 2        ;BIT 1
   .BINAER 4    .BINAER 4     .BINAER 4     .BINAER 4
   0101         0100          0010          0001
   0110         0110          0011          0011
   0111         0111          0101          0101
   1000         1000          1000          0111
   1001         1001          1001          1001
   .PSEUDO      .PSEUDO       .PSEUDO       .PSEUDO
   1010         1010          1010          1010
   1011         1011          1011          1011
   1100         1100          1100          1100
   1101         1101          1101          1101
   1110         1110          1110          1110
   1111         1111          1111          1111
   .ENDE        .ENDE         .ENDE         .ENDE
```

Jeder dieser Eingabetexte wird in einer Datei (File) unter einem anderen Namen abgespeichert.

Problemlösung

Ist die Eingabe ohne Syntaxfehler, so beginnt der LOGASM mit dem Minimiervorgang. Nach Beendigung wird die Lösung in folgender Form auf den Bildschirm, den Drucker bzw. einen Datenträger (Floppy-Disk) geschrieben:

4	3	2	1
= x1x1	= x1x0	= x01x	= xxx1
+ x11x	+ x11x	+ x101	
+ 1xxx	+ 1xxx	+ 1xxx	

Dieser Ausdruck am Terminal stellt die minimale Form dar. In eine logische Gleichung gebracht, bedeutet dies für

```
AUSGANGSBIT 4:    A4 =   E3 * E1
                   +     E3 * E2
                   +     E4

AUSGANGSBIT 3:    A3 =   E3 * /E1
                   +     E3 * E2
                   +     E4

AUSGANGSBIT 2:    A2 =   /E3 * E2
                   +     E3 * /E2 * E1
                   +     E4

AUSGANGSBIT 1:    A1 =   E1
```

wobei das Zeichen „+" die Logikfunktion ODER
„*" die Logikfunktion UND
„/" die Logikfunktion NICHT
darstellt.

Mit diesen vier Gleichungen läßt sich ein PAL programmieren. Man erhält dadurch einen Umcodierer von BCD-Code in AIKEN-Code, also die gesuchte Problemlösung.

Die oben angeführten vier Eingabetexte kann man auch in dezimaler bzw. hexadezimaler Form eingeben.

Für Bit 2 sieht das so aus:

```
.TITEL: BCD-AIKEN CONVERTER, BIT 2
.LIST :CO:

;BEZEICHNUNG DES EINGABEFORMATS
;UND DER BITBREITE

.DEZIMAL 4
2
3
5
8
9

.PSEUDO
.HEXADEZIMAL
A
B
C
D
E
F

.ENDE
```

Makro-Befehle

Das Programm stellt einen Makrobefehl zur Verfügung, der das mühselige Eintippen von Zahlensequenzen erspart. Ununterbrochen fortlaufende Zahlenkolonnen können auch als Bereich angegeben werden. Der Befehl besteht aus dem Anfangswert, dem Endwert, getrennt durch das Zeichen „–". So erzeugt z. B. der Befehl „8–11" die Zahlenfolge 8, 9, 10 und 11. Damit sind große Statusmaschinen mit nur wenigen Zeilen definierbar, und die Fehlerwahrscheinlichkeit beim Eintippen der Eingangssequenz ist minimal.

Anhand eines Beispiels lassen sich die Vorteile des Makrobefehls leicht aufzeigen:

Eingabeformat ohne Makro	Eingabeformat mit Makro
;BIT 4	;BIT 4
.BINAER 4	.BINAER 4
0000	0000–0110
0001	1111
0010	.ENDE
0011	
0100	
0101	
0110	
1111	
.ENDE	

Solche Makrostrukturen können entsprechend dem binären Format auch in hexadezimaler oder dezimaler Form erfolgen:

```
.DEZIMAL 4          .HEXADEZIMAL 4
0–6                 0–6
15                  F
.ENDE               .ENDE
```

Noch eine Aufgabe

Eine Zustandsmaschine, ein zyklischer Zähler mit einer Bitbreite von 9, soll realisiert werden. Hier handelt es sich um ein sequentielles Beispiel. Das bedeutet, daß das Bitmuster, das momentan am Ausgang der Schaltung anliegt, der Eingangsterm ist, der für den nächsten Zählvorgang verantwortlich ist (eine sogenannte Rückkopplung).

Die Zustandstabelle wird in ähnlicher Weise wie die Tabelle aus dem ersten Beispiel erstellt. Man schreibt in den linken Teil der Zustandstabelle (Eingangsterme) das erste Bitmuster: „0 0000 0000". In den rechten Teil (Ausgangsterme) notiert man das darauffolgende Bitmuster: „0 0000 0001". Es ist zugleich der Eingangsterm des nächsten Zählvorganges. Darum wird der Term „0 0000 0001" auch in den linken Teil der 2. Zeile geschrieben. Dieser Zyklus wiederholt sich bis zum Ein-

gangsterm „1 1111 1111“, der den Ausgangsterm „0 0000 0000“ zur Folge hat und damit den Zähler von vorne startet.

Die komplette Tabelle von $2^9 = 512$ Zuständen anzuführen, würde zu weit führen, außerdem ist es auch nicht Sinn des Miniervorgangs, mit Hilfe des LOGASM lange Zustandstabellen aufzustellen.

Mit den Makrobefehlen im LOGASM-Programm können auch lange Wertetabellen in ein komprimiertes Format gebracht werden. Nachfolgendes Beispiel des Eingabetextes für das MSB eines 9-Bit-Zählers zeigt, daß sich mit nur wenigen Definitionszeilen eine solche Funktion beschreiben läßt.

```
.TITEL: 9-BIT ZAEHLER (MSB) Q8
.LIST :CO:
;AUSGAENGE WERDEN AUF "0" MINIMIERT
.DEZIMAL 9
0–254
511
;PSEUDOTERME KOMMEN IN EINER
;ZAEHLERFUNKTION NICHT VOR!
.ENDE
```

Als Resultat nach dem LOGASM-Durchlauf erscheint am Bildschirm:

```
LOESUNG = 1.1111.1111     /Q8:= Q8* Q7*Q6*Q5*Q4*Q3*Q2*Q1*Q0
        + 0.0XXX.XXXX         + /Q8*/Q7
        + 0.X0XX.XXXX         + /Q8*/Q6
        + 0.XX0X.XXXX         + /Q8*/Q5
        + 0.XXX0.XXXX         + /Q8*/Q4
        + 0.XXXX.0XXX         + /Q8*/Q3
        + 0.XXXX.X0XX         + /Q8*/Q2
        + 0.XXXX.XX0X         + /Q8*/Q1
        + 0.XXXX.XXX0         + /Q8*/Q0
```

Fazit

Der LOGASM ist besonders für die Erstellung logischer Gleichungen zur Programmierung von PAL-Bausteinen konzipiert, was natürlich andere Anwendungen nicht ausschließt.

Das Programm übersetzt jede beliebige Zustandstabelle mit bis zu 15 Eingangsvariablen in ein Gleichungsformat, das sofort vom PALASM (PAL-Assembler) übernommen werden kann.

Damit liefert LOGASM von einfachen Decodieraufgaben bis zu extensiven Statusmaschinen sofort die dementsprechenden Gleichungen, ohne in die Gatterebene zurückkehren zu müssen.

Gisbert Vollmer

PROMs und PALs mit PC entwickelt und getestet

Zwar ist es prinzipiell möglich, Festwertspeicher und PALs „zu Fuß“ zu programmieren; auf die Dauer wird dies jedoch reichlich mühselig. Tatsächlich geht es hierbei ja vor allem um das an sich rein mechanische Umsetzen logischer Funktionstafeln in einen Plan, der die zu unterbrechenden Verbindungen enthält. Und das ist eine klassische Aufgabe für einen Rechner, vor allem, wenn es sich um größere Stückzahlen handelt. Er kann darüber hinaus auch gleich die fertigen Bauelemente prüfen. Welche Möglichkeiten derzeit bereits zur Verfügung stehen und welche Anforderungen ein Personal Computer erfüllen muß, sei anschließend diskutiert.

PROM-Programmierung

Ein Programmiergerät, das an einen Personal Computer angeschlossen wird, muß die serielle Schnittstelle (RS-232 C) und die Firmware mit Standard Remote Control (SRC) oder Computer Remote Control (CRC) besitzen. SRC bedeutet, daß ein Gerät mit einfachen Syntax-Befehlen, die auch das Gerät auf der Tastatur besitzt, steuerbar ist. CRC erfordert eine auf einem Rechner lauffähige sogenannte Treibersoftware, so daß das Programmiergerät mit verkürzten Befehlen vom Entwicklungssystem aus fernsteuerbar ist. Mit dem Personal Computer ist je nach Software beides im Prinzip möglich.

Ist nun der Personal Computer – zum Beispiel ein Compaq oder IBM – mit einer Quadram-Karte (2 × RS-232 C und zusätzlich 256 KByte RAM) ausgerüstet, wird der Programmer mit einem RS-232C-Kabel angeschlossen. Vom Personal Computer werden die Schnittstellen mit „COM 1“ und „COM 2“ angesprochen. Minimum der Verkabelung ist ein „Null-Modem“, bei dem der Anschluß 1 direkt, die Anschlüsse 2 und 3 jedoch über Kreuz miteinander verbunden sind. Der Befehl, das Programmiergerät in SRC zu bringen, ist bei Data I/O z. B. „Select FB“. Der Personal Computer wurde vorher mit der Software „VTERM“ in den Status des Terminals VT-100 gebracht.

Nun wird das Programmiergerät nur noch über die gewohnte Tastatur gesteuert. Aufgrund der vollen Editierfähigkeit des Programmers läßt sich das RAM auf dem Bildschirm in sedezimaler, oktaler oder binärer Darstellung sehr übersichtlich zur Anzeige bringen. Änderungen sind ebenso deutlich und einfach vorzunehmen.

Die gesamte Palette der „Select“-Funktion erscheint gemeinsam auf dem Bildschirm. Der Anwender kann bequem – auch aus den zahlreichen Untermenüs – wäh-

```
A B E L (tm) -- COPYRIGHT (c) 1983,1984 DATA I/O CORP., REDMOND, WA   page 1

 BASIC LOGIC DESIGN    Basic logic source file                        page 2

 The source file below describes our logic design in ABEL.  Writing the design
 is a straightforward process of declaring the target device, naming the input
 and output pins, writing the logic equations, and writing the test vectors.
 Read through the design here to get a feel for design expression with ABEL, and
 the details will be covered on the next few pages.

 The line numbers on the right are not part of a real ABEL source file; they are
 only used for reference further in this lesson.

              -- BASIC_LOGIC SOURCE FILE (scroll to view) --
 -------------------------------------------------------------------------------

                                                                    (Line nos.)
module  BASIC_LOGIC;                                                    (1)

title                                                                   (2)
'ABEL basic logic design example       11 Nov. 1983                     (3)
Data I/O Corp.   Redmond, WA'                                           (4)

" You can place comments anywhere, after a quotation mark.

    IC4                   device  'P10L8';            " declare device     (5)

"
" Declare our input pins:
"
    NA1, NA2, NA3         pin      1,2,3;             " NAND inputs        (6)
    AOI1, AOI2,                                                            (7)
    AOI3, AOI4            pin      4,5,6,7;           " AND_OR_INV inputs  (8)
    X1, X2                pin      8,9;               " XOR inputs         (9)
"
" Declare output pins:
"
    NAND,                                                                  (10)
    AND_OR_INV, XOR       pin      19,18,17;                               (11)
"
" Logic description by Boolean equations:
"
equations                                                                  (12)
    NAND         = !(NA1 & NA2 & NA3);                                     (13)
    AND_OR_INV   = !((AOI1 & AOI2) # (AOI3 & AOI4));                       (14)
    XOR          = X1 $ X2;                                                (15)
"
" Test vectors for NAND gate only:
"
test_vectors    ( [NA1, NA2, NA3] -> NAND)                                 (16)
    [0, 0, 0]  ->  1;                                                      (17)
    [0, 0, 1]  ->  1;                                                      (18)
    [0, 1, 0]  ->  1;                                                      (19)
    [0, 1, 1]  ->  1;        " test vectors are shown as                   (20)
    [1, 0, 0]  ->  1;        " inputs -> outputs                           (21)
    [1, 0, 1]  ->  1;                                                      (22)
    [1, 1, 0]  ->  1;                                                      (23)
    [1, 1, 1]  ->  0;                                                      (24)
```

len. Diese Funktionen betreffen RAM-Änderungen, Schnittstellen-Einstellungen, Wahl der Daten-Übertragungsformate, Kalibrierung und Editierung. Die in diesem Beitrag gezeigten Listings sind typische Beispiele und sprechen für sich.

Sehr vorteilhaft wirkt sich diese Anwendung bei den großen EPROMs (bis 27512 von AMD und Intel) aus. Auch die Freigabe aller „schnellen Algorithmen" bei Data I/O und die „Silicon Signatures" kommen dem

```
A B E L (tm) -- COPYRIGHT (c) 1983,1984 DATA I/O CORP., REDMOND, WA   page 1

 AUTO ALARM DESIGN     Examining the simulation file                 page 19

 A partial listing of the simulation file ALARM.SIM is shown below.  The format
 is similar to that for the Basic Logic design example, only now a "## Clock ##"
 line is shown to indicate the transition from the current state to the next
 state.

            -- SIMULATION FILE, TRACE LEVEL 1 (scroll to view) --
 ------------------------------------------------------------------------------

Simulate device U1, type 'P16R4'

Vector 1
Output   [                    ] -> [..........ZLLLLLHH.]
## Clock ##
Output   [                    ] -> [..........ZLHHHHHH.]
V 0001   [C0100000000000000000] -> [..........NLHHHHHH.]

Vector 2
Output   [                    ] -> [..........ZLHHHHHH.]
## Clock ##
Output   [                    ] -> [..........ZLLHHHHH.]
V 0002   [C1000000000000000000] -> [..........NLLHHHHH.]

Vector 3
Output   [                    ] -> [..........ZLLHHHHH.]
## Clock ##
Output   [                    ] -> [..........ZLLLHHHL.]
V 0003   [C0011000000000000000] -> [..........NLLLHHHL.]

Vector 4
Output   [                    ] -> [..........ZLLLHHHL.]
## Clock ##
Output   [                    ] -> [..........ZLHHHHHH.]
V 0004   [C0100000000000000000] -> [..........NLHHHHHH.]
                              .
                              .
                              .
 ---------------------------------------------------------------------------

                 U or PgUp: scroll up     D or PgDn: scroll down H
 Space: next page    B: back up   G: go to page    P: print    R: return to menu
```

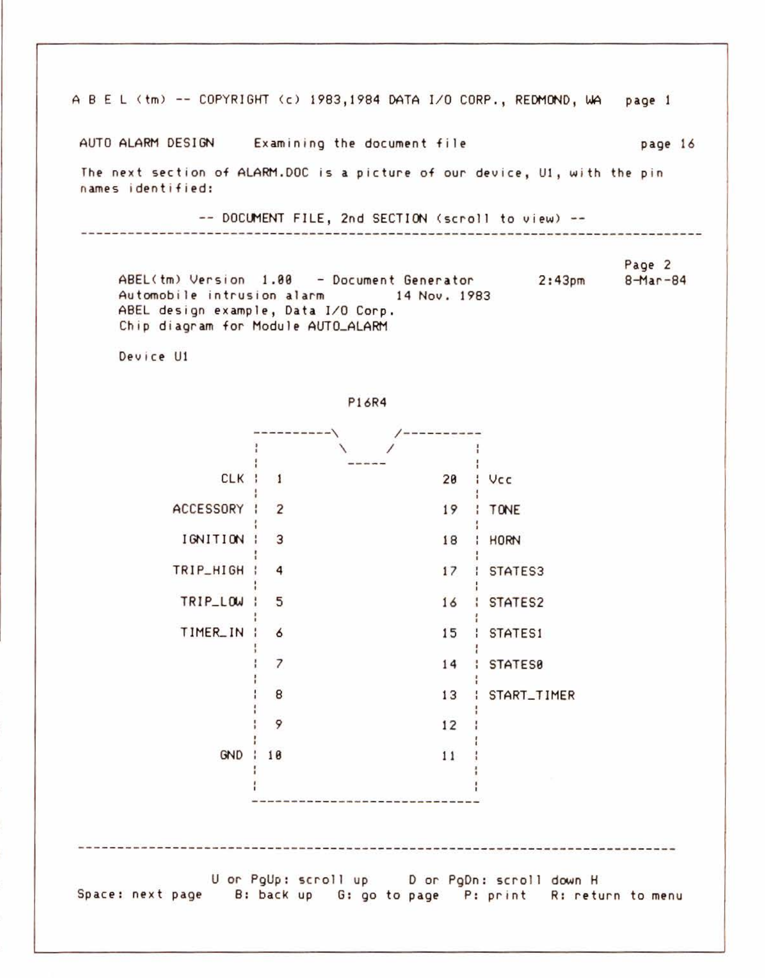
Anwender hier zu Hilfe. Der Personal Computer bietet darüber hinaus gleich genügend Speicherplatz, um die „Set-Programmierung", bei der mehrere Bausteine verschiedenen Inhalts gleichzeitig geladen und programmiert werden, großzügig handhaben. Schließlich lassen sich alle Daten mit dem Personal Computer auf einfache Weise auf Disketten oder über den parallel angeschlossenen Drucker dokumentieren. Wegen seiner Universalität und zunehmender Präsenz können solche Dokumente flexibler an andere Einsatzorte gebracht werden.

Neuerdings gibt es eine Software, die auf Personal Computern mit dem Betriebssystem MS-DOS (CP/M in Vorbereitung) läuft. Dieser, PROMLINK genannte, Software-Treiber kann menügeführt und interaktiv nicht nur Files übertragen, sondern führt auch alle sonstigen notwendigen Kommandos wie z. B. „Load", „Verify", „Program", „Select" und „I/O Format" aus. Fehlermeldungen erscheinen im Klartext; da PROMLINK einen umfangreichen Katalog an Standardbauelementen enthält, muß man nur noch den jeweiligen Typ angeben – der Rest läuft praktisch automatisch ab.

Entwicklung und Programmierung von PALs und IFLs

Gerade bei der Entwicklung von PALs, IFLs und allen weiteren logischen Bauelementen rückt der Personal Computer immer stärker in den Vordergrund. Um ein PAL (Programmable Array Logic) oder IFL (Integrated Fuse Logic) zu entwickeln, bedarf es einer Software, die es gestattet, bestimmte logische Verknüpfungen in ein Bitmuster umzusetzen.

Bei einem PAL handelt es sich bekanntlich um einen Baustein, der ein frei programmierbares Eingangs-Array und ein fest konfiguriertes Ausgangs-Array besitzt. Das Ausgangsfeld enthält die logischen Verknüpfungen z. T. mit Rückführungen, Registern, programmierbarer Ausgangspolarität usw. zu den Ausgangsanschlüssen des



```
A B E L (tm) -- COPYRIGHT (c) 1983,1984 DATA I/O CORP., REDMOND, WA   page 1

 AUTO ALARM DESIGN     Examining the document file                 page 16

 The next section of ALARM.DOC is a picture of our device, U1, with the pin
 names identified:

            -- DOCUMENT FILE, 2nd SECTION (scroll to view) --
 ------------------------------------------------------------------------------

                                                                 Page 2
      ABEL(tm) Version  1.00   - Document Generator      2:43pm    8-Mar-84
      Automobile intrusion alarm        14 Nov. 1983
      ABEL design example, Data I/O Corp.
      Chip diagram for Module AUTO_ALARM

      Device U1

                              P16R4

                     ----------\     /----------
                     !          \   /          !
                     !           ---           !
                 CLK ! 1                    20 ! Vcc
                     !                         !
           ACCESSORY ! 2                    19 ! TONE
                     !                         !
            IGNITION ! 3                    18 ! HORN
                     !                         !
           TRIP_HIGH ! 4                    17 ! STATES3
                     !                         !
            TRIP_LOW ! 5                    16 ! STATES2
                     !                         !
            TIMER_IN ! 6                    15 ! STATES1
                     !                         !
                     ! 7                    14 ! STATES0
                     !                         !
                     ! 8                    13 ! START_TIMER
                     !                         !
                     ! 9                    12 !
                     !                         !
                 GND ! 10                   11 !
                     !                         !
                     -------------------------

 ---------------------------------------------------------------------------

                 U or PgUp: scroll up     D or PgDn: scroll down H
 Space: next page    B: back up   G: go to page    P: print    R: return to menu
```

PALs. Die IFLs bestehen dagegen aus zwei oder mehr frei programmierbaren Feldern, so daß bei Ihnen eine größere Flexibilität in der Anwendung zur Verfügung steht.

Die Hersteller programmierbarer Logikbausteine bieten i. a. passende Software-Pakete an, die auf diversen Rechnern bzw. Entwicklungssystemen laufen. Am weitesten verbreitet dürfte hier PALASM („PAL ASseMbler") von MMI sein. Hierfür – und auch für IFLs – gibt es zu den Programmiergeräten von Data I/O einen Hard-

ware-Adapter, der die Verbindung zum Terminal herstellt.

Die Source-Daten bestehen aus Boole'schen Gleichungen. Diese beziehen sich in ihrer Bezeichnung und Zuordnung auf die Anschlüsse und inneren Zusammenhänge des PAL. Ist alles logisch miteinander verknüpft, so kann die Assemblierung erfolgen. Das „Fuse Array" entsteht und ist entsprechend dem PAL-Typ verschieden breit und lang. Der PALASM gibt in Klartext Fehlermeldungen aus und erlaubt Änderungen in den Boole'schen Gleichungen oder im Fuse-Array.

Ein Personal Computer allerdings bringt weitere Vorteile gegenüber einem gewöhnlichen Terminal. Die Schnittstelle (RS-232C) ist softwaremäßig vom Personal Computer aus einstellbar. Alle entwickelten Daten können auf Disketten dokumentiert werden.

Das PALASM-Programm bietet bis heute eine Entwicklungs-Unterstützung, die in fast allen Fällen ausreicht. Es ist darüber hinaus kostengünstig und in der Anwendung durch Menüführung und feste Tastenfunktionen einfach zu handhaben. Seine Nachteile liegen darin, daß es durch das Source-Programm (FORTRAN) in der Gestaltung festgelegt ist; bestimmte Eingaben sind also an genau definierten Stellen vorzunehmen.

Einen Ausweg aus diesem Dilemma bietet die von Data I/O entwickelte Hochsprache „abel" (Advanced Boolean Expression Language) zur Entwicklung aller logischen Bauelemente. Diese komfortable Software läuft auf DEC-Maschinen sowie auf allen PC-DOS-Systemen. Abel ist in „C" geschrieben und gestattet eine höchst flexible Arbeitsweise. Über einen Editor (z. B. „Quicksoft") kann das Design in Boole'schen Gleichungen – Sets, Status-Diagrammen oder Wahrheitstabellen – eingegeben werden. Dabei bleibt es dem Anwender frei überlassen, wie er seine Source-Daten konfiguriert. Lediglich die „Modul"- und „Device"-Bezeichnung sind einzugeben, ebenso wie die Key-Worte „equations", „testvectors" und „end". Es gibt die Möglichkeit, mit „Makros" oder auch „Nodes" zu arbeiten, d. h. gewisse Sachverhalte sinnvoll zusammenzufassen und zur Verarbeitung zu bringen. Die Anschlußbezeichnung sowie Kommentare sind frei einsetzbar.

Eine kennzeichnende Eigenschaft von „abel" ist die schrittweise Reduzierung und Optimierung des Entwurfs durch volle Ausnutzung des Rechenpotentials, das der Personal Computer bietet.

Nachdem die logischen Eingaben in Boole'sche Form umgesetzt werden, prüft „abel" jede einzelne der so entstandenen Gleichungen auf Schlüssigkeit und hinsichtlich ihrer Realisierbarkeit im gegebenen Baustein. Danach läßt sich mit Hilfe von „PRESTO"-Algorithmen der gesamte Entwurf, also das Zusammenspiel aller Gleichungen miteinander, prüfen und eventuell weiter optimieren.

```
A B E L (tm) -- COPYRIGHT (c) 1983,1984 DATA I/O CORP., REDMOND, WA   page 1

AUTO ALARM DESIGN     Examining the document file                    page 17

The fuse map section of the document file contains fuse maps for each device in
the design.  A fuse map is a graphical representation of the programmed fuse
array for the device, arranged to resemble the device logic diagram provided by
the device manufacturers.  First fuse numbers and a break every tenth fuse are
shown for calculation of specific fuse numbers.  The fuse map section of
ALARM.DOC requires two pages, and is shown below.

              -- DOCUMENT FILE, 3rd SECTION (scroll to view) --
--------------------------------------------------------------------------------

                                                                  Page 3
     ABEL(tm) Version  1.00   - Document Generator        2:43pm   8-Mar-84
     Automobile intrusion alarm          14 Nov. 1983
     ABEL design example, Data I/O Corp.
     Fuse Map for Module AUTO_ALARM

     Device U1

           0          10         20         30
     0: ---------- ---------- ---------- --      1152: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
    32: ---------- X----X---X --X------- --      1184: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
    64: ---------- X---X----X ---X------ --      1216: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
    96: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1248: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   128: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1280: -----X---- X---X---X- ---X------ --
   160: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1312: X--------- X---X---X- --X------- --
   192: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1344: -----X---- X--XX----- ---X------ --
   224: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1376: -----X--X- X---X----- ---X------ --
   256: ---------- ---------- ---------- --      1408: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   288: ---------- X----X--X- --X------- --      1440: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   320: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1472: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   352: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1504: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   384: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1536: ---------- ---------- ---------- --
   416: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1568: ---------- X---X----- ---------- --
   448: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1600: ---------- X----X--X- --X------- --
   480: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1632: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   512: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1664: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   544: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1696: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   576: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1728: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   608: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1760: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   640: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1792: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   672: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1824: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   704: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1856: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   736: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      1888: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   768: -----X---- X--X-----X --X------- --      1920: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   800: -----X--X- X--------X --X------- --      1952: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   832: -----X---- X----X---- --X------- --      1984: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   864: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX      2016: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   896: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   928: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   960: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
   992: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  1024: -----X---- X---X----X ---------- --
  1056: -----X---- X--XX----- ---X------ --
  1088: -----X--X- X---X----- ---X------ --
  1120: -----X---- X-----X--X --X------- --
```

Selbst ein noch wenig erfahrener Anwender dürfte kaum Probleme beim Arbeiten mit „abel" bekommen. Hierfür sorgen allein schon die sehr ausführlichen Fehlermeldungen, die auf Verletzungen von Entwurfsregeln aufmerksam machen.

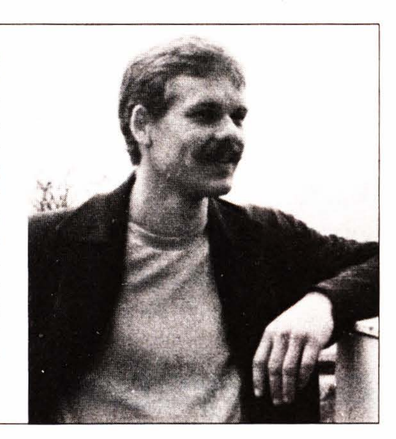
Im letzten Teil der Bearbeitung eines Designs durch „abel" wird ein Dokumentationsfile mit Source-Daten, Fuse-Map und Layout des Bausteins erzeugt. Dieses läßt sich auf Diskette speichern sowie über den Drucker am Personal Computer ausgeben.

Testen logischer Entwicklungen

Ein sehr wichtiger Teil des Testens von logischen Entwicklungen einerseits und des Testens der Bausteine selbst soll in diesem Abschnitt extra angesprochen werden. Die Testdaten sind sowohl beim PALASM als auch bei „abel" Bestandteil der angesprochenen Source-Daten und gehören zu jeder Dokumentation.

Aufgrund des Aufbaus eines PALs und auch eines IFLs leuchtet es sofort ein, daß ein einfaches Testen des Fuse-Arrays, also der assemblierten Daten, nicht mehr ausreicht. Bei PROMs genügt der Verify-Vorgang mit V_{CC} High und V_{CC} Low. Das wird zwar entsprechend bei

Dipl.-Ing. Gisbert Vollmer ist in Hamburg geboren und aufgewachsen. Im benachbarten Wedel/Holstein studierte er Physikalische Technik. Nach einem halbjährigen Aufenthalt in Kanada fand er bei der Firma Instrumatic Electronic GmbH in München den passenden Arbeitsplatz. Als Produktspezialist unterstützte er dort von 1983 bis Anfang 1985 den Vertrieb von elektronischen Meßsystemen. Der Autor ist inzwischen selbst Unternehmer und bietet einen Programmierservice für PROMs, PALs und IFLs sowie Unterstützung im CAE-Bereich an.



logischen Bausteinen auch vorgenommen, reicht aber nicht aus; denn logische Verbindungen von Eingängen zu Ausgängen müssen auf ihre Realisierung durch den Baustein getestet werden. Dazu gibt man ein zweites Mal bestimmte Bestandteile des Designs in Form einer Funktions-Tabelle (PALASM) oder, sehr viel freier, mit Struktur-Testvektoren (abel) ein.

Hier zeigt sich wiederum ganz deutlich der Vorteil von „abel" in Verbindung mit einem Personal Computer. Um spätere Enttäuschungen und die Kosten für eine aufwendige Nacharbeit zu vermeiden, lohnt sich die zusätzliche Mühe, den Entwurf nochmals überprüfen zu lassen. Die Flexibilität dieses Software-Paketes auch beim Test kommt dem natürlichen Sicherheitsdenken des Anwenders entgegen. Tatsächlich besteht die abschließende Simulation aus einem vollständigen Vergleich der Vorgaben – seien es Wahrheitstabellen oder Statusbedingungen – mit den schließlich assemblierten Daten des Bauelementes. Gleichzeitig generiert „abel" die Testvektoren für die später in der Produktion durchgeführten Hardware-Prüfungen; sie sind Teil der Dokumentation.

Die Struktur-Testvektoren gehören auch zum JEDEC-Format, das internationale Standardformat für alle programmierbaren Logik-Bausteine. Wenn ein solcher Baustein von der Entwicklung zur Fertigung geht, sind im dazugehörigen JEDEC-Format das Fuse-Array (mit 0 und 1) sowie die Test-Bedingungen enthalten.

Die Dokumentation dieses Formates ist auf dem Personal Computer sehr einfach. Ein von „abel" erzeugter JEDEC-File für jeden Baustein wird auf Diskette in die Produktion geliefert, wo dieser File sehr leicht über eine RS-232C-Schnittstelle in das Programmiergerät übertragbar ist. Die Handhabung des Personal Computers ist in beiden Abteilungen dabei die gleiche.

Bei der Anfertigung von Duplikaten bietet sich neben den Struktur-Testvektoren die Möglichkeit des Fingerprint-Tests an. Hierbei handelt es sich um ein auf der Signatur-Analyse basierendes Pseudorandom-Testverfahren. Nach einem beliebig vorgegebenen „Startvektor" erzeugt hierbei ein Schieberegister 128 000 weitere Testzustände. Das Ergebnis ist eine aus jedem Zustand aufaddierte achtstellige Sedezimalzahl, die typisch für ein bestimmtes Design mit der Startbedingung ist und „Fingerprint" heißt. Jeder weitere produzierte Baustein besitzt diesen charakteristischen Fingerabdruck.

Mit der „Last Fuse" oder „Security Fuse" kann in einem Master-Baustein ein nachträgliches Auslesen – nicht aber das logische Testen – unmöglich gemacht werden. Als Neuheiten gibt es von AMD und MMI Register-PALs mit programmierbarer Ausgangspolarität und von Harris CMOS-PALs. National bietet „A"-Versionen mit hoher Geschwindigkeit. Außerdem kann mit der Preload-Option in Testvektoren ein „P" eingegeben werden, um die Register-PALs im eindeutigen Zustand zu testen. Diese Möglichkeiten sowie die soft- und hardwaremäßigen Unterstützungen aller neuen Bausteine von AMD, Harris, MMI, National, Texas Instruments und Signetics sind von Data I/O bereits realisierbar. Hinzu kommt, daß „abel" die erste Software ist, mit der der 22V10 von AMD und der 32R16, Bausteine großer Flexibilität, entwickelbar sind.

Je umfangreicher und komplexer die Logik-Bausteine werden, um so flexibler und übersichtlicher muß eine Entwicklungssprache sein, und um so leichter muß sie gehandhabt werden können. Auch hier besitzt der Personal Computer die eingangs erwähnten Vorteile.

Willibald Voldan

Makro-PAL-Assembler zur Systemerstellung

Die Schaltungsentwicklung mit programmierbarer Logik stellt eine besondere Herausforderung für den Designer digitaler Schaltungen dar. Die schnellen Entwicklungszeiten mit diesen Bauelementen werden von einem Softwarepaket (PALASM-II) zur Logikerstellung unterstützt, das die hohe Komplexität der MegaPALs mit einer strukturierten Programmsprache effizient ausnützt. Dieses Programm liest eine PAL-Schaltungsspezifikation in Form Boole'scher Gleichungen oder Zustandstabellen und produziert daraus den dementsprechenden Sicherungsplot (die der logischen Funktion entsprechende interne Verdrahtung des PALs) und das Programmierformat (JEDEC-Format).

Ein zustandsbetriebener Simulator, der in seiner Definitionssprache an eine der höheren Programmiersprachen in der Software angelehnt ist (Pascal), simuliert die Eingabe der Gleichungen und produziert daraus Testvektoren. Der PALASM-II, das Nachfolgeprogramm des bekannten PALASM, akzeptiert die Definition von Makros in der Anschlußbelegung, den logischen Gleichungen und der Funktionsbeschreibung des Simulators. Um die Eingabe der logischen Gleichungen noch effizienter zu gestalten, ist auch die Verwendung einmal festgelegter String-Sequenzen möglich.

In einer weiteren Ausbaustufe dieses Programms werden dann auch automatisch Testvektoren erstellt und die Eingabe bei Bedarf minimiert.

Das Programm handhabt alle heute von Monolithic Memories verfügbaren PAL-Bausteine im 20-, 24-, 40- und 84-Pin-Gehäuse (Standard-, RA-, RS-, MegaPALs) und ist für zukünftige Typen leicht erweiterbar. PALASM-II kann sogar mehrere PAL-Bausteine in einem Eingabeformat bearbeiten. Damit lassen sich ganze Systeme auf der Basis Boole'scher Gleichungen definieren und simulieren.

Nachdem Kommentare wie Titel, Autor, Datum, Patternnummer, Revisionsstand usw. eingegeben wurden, beginnt das eigentliche Programm mit der Definition der Pinbelegung. Jeder Pinname kann dabei bis zu 20 ASCII-Zeichen aufweisen. Mit der Verwendung von Makro-Definitionen wird einiges an Zeit und Arbeit eingespart; die Sequenz

```
B {7...0}
```

steht an Stelle von

```
B{7} B{6} B{5} B{4} B{3} B{2} B{1} B{0}
```

Kommen in den Gleichungen immer wiederkehrende Ausdrücke unverändert vor, ist es sinnvoll, diese über String-Sequenzen einmal einem Namen zuzuordnen, der diese Verknüpfung repräsentiert:

```
STRING ENABLE '/B{0} * B{1} * /B{2}'
```

Danach wird immer, wenn das Wort „ENABLE" im Eingabeprogramm vorkommt, dafür das Produkt der drei oben angeführten Pins in den Gleichungen eingesetzt. Auch bereits reservierte Operanden wie AND, OR und BIN (binäre Decodierung) sind in den Gleichungen erlaubt. Mit Hilfe von zweidimensionalen Vektoren können auch Makros in den Boole'schen Gleichungen verwendet werden.

Die Funktion

```
Y {I=3...0} = A{I} * S + B{I} * /S
```

ersetzt vier Gleichungen

```
Y0 = A0 * S + B0 * /S
Y1 = A1 * S + B1 * /S
Y2 = A2 * S + B2 * /S
Y3 = A3 * S + B3 * /S
```

Damit lassen sich große Sequenzen, wie z. B. ein 16-Bit-Zähler, mit wenigen Gleichungen definieren:

```
STRING COUNT '/LOAD * /CLEAR'        ;STRING

Q {I=0...15} := Q{I} * COUNT          ; COUNT
              + D{I} * LOAD * /CLEAR  ; LOAD
              :+: Q{I-1} *...* Q{0} * CIN * COUNT
                                      ; HOLD

COUT = Q{15} *...* Q{0} * CIN         ; CARRY OUT
```

Nachdem die Schaltung mit logischen Gleichungen beschrieben ist, kann die Simulation mit strukturierten Steuerbefehlen wie SET, CLOCK, CHECK, FOR, IF

THEN ELSE, BEGIN, ELSE, TRACE_ON, TRACE_OFF, um nur einige zu nennen, die Schaltung überprüfen. Dabei werden die Eingänge auf bestimmte logische Pegel gesetzt und die Ausgänge auf vorgegebene Pegel überprüft.
Die Instruktion
SET /ENA, XYZ, /STROBE
bewirkt, daß
die Eingänge ENA und STROBE auf LOW sowie
der Eingang XYZ auf HIGH gesetzt wird.

Beispiel einer Schaltungsdefinition für den PALASM-II

```
TITLE     32 BIT ADDRESSABLE REGISTER
          DESIGN SPECIFICATION
PATTERN   EXAMPLE PATTERN FOR PALASM-II
          SOFTWARE
REVISION
AUTHOR    JERRY GREINER, WILLY VOLDAN
COMPANY   MONOLITHIC MEMORIES GMBH,
          MUNICH
DATE      03/03/1985

CHIP1 THIRTY_TWO_BIT_ADDRESSABLE_REGISTER
PAL32R16
Q{0...3} /E2 NC NC A{0...1} VCC A{2...4} DATA
/PRLD1 CLK1 Q{4...11}
/E1 NC NC NC NC NC GND NC NC NC /PRLD2 CLK2
Q{12...15}

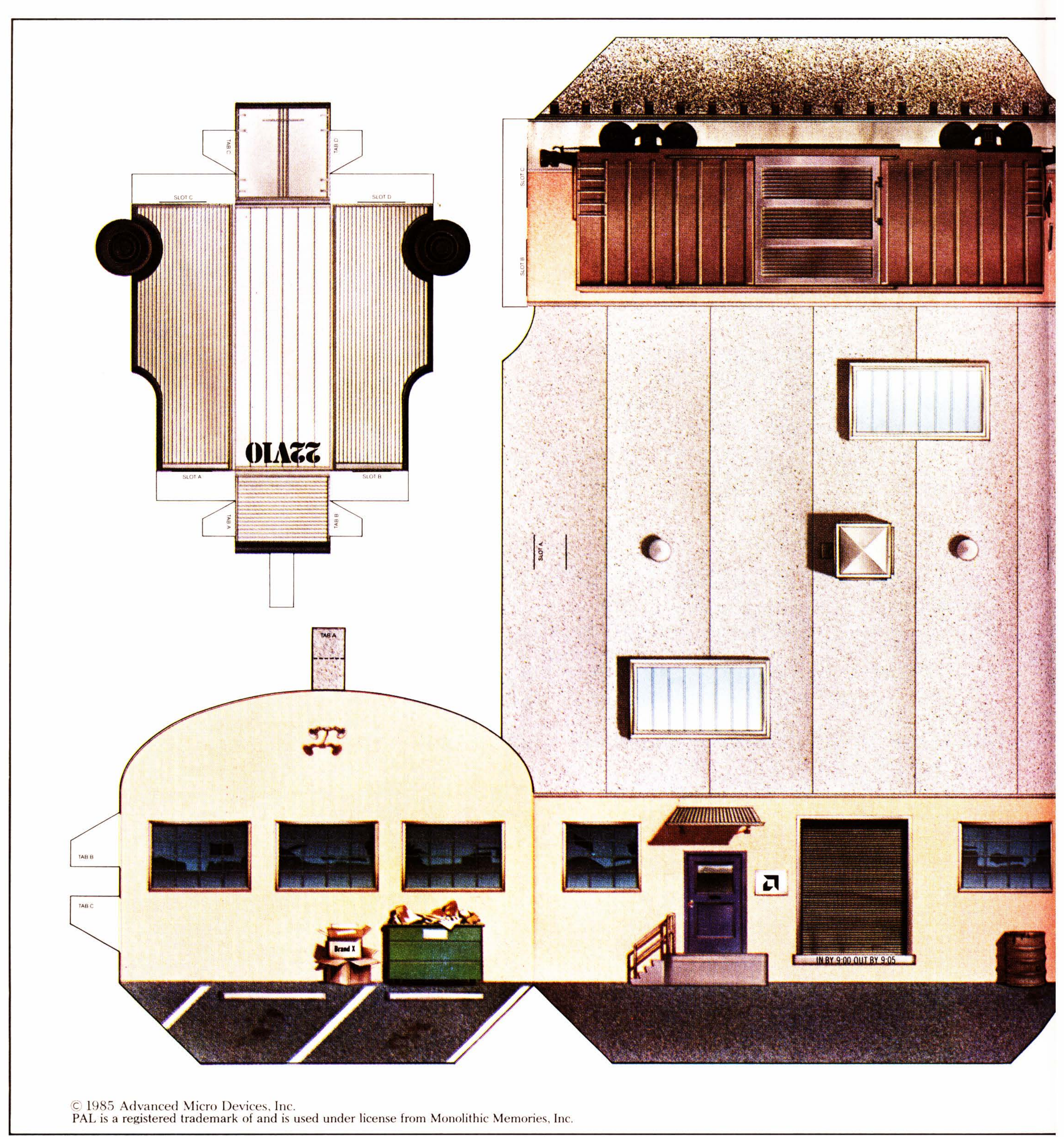
CHIP2 THIRTY_TWO_BIT_ADDRESSABLE_REGISTER
PAL32R16
Q{16...19} /E4 NC NC A{0...1} VCC A{2...4} DATA
/PRLD3 CLK3 Q{20...27}
/E3 NC NC NC NC NC GND NC NC NC /PRLD4 CLK4
Q{28...31}

EQUATIONS
STRING DECODE 'BIN{I,J=4...0} {ADDR{J}}'
                                  ; ADDRESS DECODE

Q{I=31...0}:=/DECODE * Q{I}      ; HOLD
          + DECODE * DATA        ; LOAD

SIMULATION ; SET ALL OUTPUTS HIGH, THEN LOW
TRACE_ON Q{31...0}
SET /Q{31...0}, A{4...0}
FOR H=1 TO 2
 BEGIN                           SET DATA
  FOR I=1 TO 2
   BEGIN                         SET /A{3...0}
    FOR J=1 TO 2
     BEGIN                       SET /A{2...0}
      FOR K=1 TO 2
       BEGIN                     SET /A{1...0}
        FOR L=1 TO 2
         BEGIN                   SET /A{0}
          FOR M=1 TO 2
           BEGIN                 CLOCK
            GENERATE
            SET A{0}
           END
         SET A{1}                END
       SET A{2}                  END
     SET A{3}                    END
   SET A{4}                      END
 SET /DATA                       END
TRACE_OFF
```

BAUEN SIE SICH



tm 4210/tm 4211

VERTRAGSHÄNDLER BUNDESREPUBLIK DEUTSCHLAND: BERLIN: **SPOERLE Electronic:** Tel.: (0 30) 6 93 40 90. DÜSSELDORF: **EBV Elektronik GmbH:** Tel.: (02 11) 8 48 46. **KONTRON Halbleiter GmbH:** Tel.: (02 11) 73 61-1 50. DORTMUND: **SPOERLE Electronic:** Tel.: (02 31) 2 18 01. DREIEICH/Ffm.: **SPOERLE Electronic:** Tel.: (0 61 03) 3 04-0. FRANKFURT: **EBV Elektronik GmbH:** Tel.: (0 69) 78 50 37. **KONTRON Halbleiter GmbH:** Tel.: (0 69) 63 17-1 44. HAMBURG: **KONTRON Halbleiter GmbH:** Tel.: (0 40) 6 82 95-1 22. **SPOERLE Electronic:** Tel.: (0 40) 5 40 41 40. HANNOVER: **KONTRON Halbleiter GmbH:** Tel.: (05 11) 83 90 51. **EBV Elektronik GmbH:** Tel.: (0 51 39) 50 38. MÜNCHEN: **ASTRONIC GmbH:** Tel.: (0 89) 30 90 31. **EBV Elektronik GmbH:** Tel.: (0 89) 61 10 51. **KONTRON Halbleiter GmbH:** Tel.: (0 89) 3 19 01-3 77. **SPOERLE Electronic:** Tel.: (0 89) 95 99 90. NÜRNBERG: **KONTRON Halbleiter GmbH:** Tel.: (09 11) 53 33 06. **SPOERLE Electronic:** Tel.: (09 11) 7 82 16. QUICKBORN: **NORDELEKTRONIK:** Vertriebs GmbH, Tel.: (0 41 06) 7 20 72. STUTTGART: **EBV Elektronik GmbH:** Tel.: (07 11) 24 74 81. **KONTRON Halbleiter GmbH:** Tel.: (07 11) 89 17-1 40. **SPOERLE Electronic:** Tel.: (0 71 42) 6 56 09.
SCHWEIZ: **KONTRON AG:** Tel.: 00 41-1-4 35 41 11.
ÖSTERREICH: **KONTRON GmbH & Co. KG:** Tel.: 00 43/2 22/67 06 31.

Willibald Voldan

Berechnung der Gatterkomplexität von PAL-Bauelementen

Zur Berechnung der Gatterkomplexität von programmierbaren Logikbausteinen definierte John Birkner, der Erfinder der PAL-Bauelemente, den Begriff „SMAC". Dieses Akronym steht für *S*tate-*M*achine-*A*tomic-*C*ell und beschreibt eine Maßeinheit zur Definition der logischen Tiefe programmierbarer Bauelemente. Vergleichbar der 1-Bit-Speicherzelle, die das Grundelement von RAMs oder PROMs darstellt, oder des 4-Eingang-NAND-Gatters, dem Grundelement der Gate-Arrays, fungiert ein SMAC als praktische Einheit zur Bestimmung der Komplexität von großen PALs, wie z. B. MegaPALs.

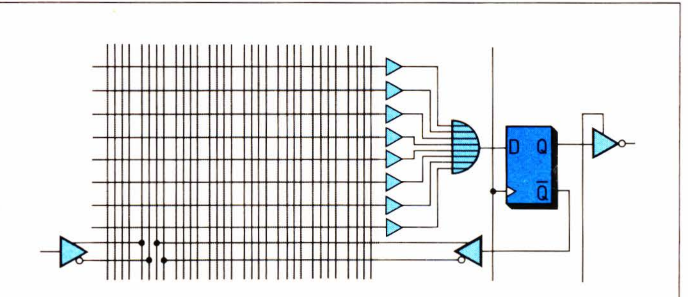


Bild 1. Das SMAC besteht aus der üblichen UND-ODER-Matrix und einem D-Flipflop. Es stellt die Bewertungsgrundlage für die Komplexität eines PALs – das „Gatteräquivalent" – dar

Bild 2. Einige typische PALs im Vergleich auf der Grundlage des SMAC und Gatteräquivalentes

	Gatter/SMAC	SMACS	Gatter gesamt mit 4 Eingängen
PAL 16R8	50	8	400
PAL 32R16	98	16	1568
PAL 64R32	178	32	5696

Ein SMAC besteht aus der bekannten UND-ODER-Struktur programmierbarer Logik, gefolgt von einem D-Register, dessen Komplementär-Ausgang /Q wieder in die programmierbare UND-Matrix rückgekoppelt ist (*Bild 1*). Diese Anordnung ist als Basiselement eine 1-Bit-Status-Maschine.

Auf Grund der mit dem Register verknüpften kombinatorischen Logik variiert die Zahl der Gatterfunktionen, aus denen ein SMAC gebildet wird, zwischen verschiedenen Anordnungen. Deshalb wird zur Berechnung der Gatterkomplexität mit SMACs von einem NAND-Gatter mit vier Eingängen ausgegangen.

Die Gleichung zur Bestimmung der Gatteräquivalenz eines SMACs lautet:

GATTER/SMAC = {GATTER/(PRODUKT-TERM)}
× (PRODUKT-TERM)
+ {GATTER/ODER-TERM}
+ (GATTER/REGISTER)
+ (AUSGANGS-GATTER)

Das MegaPAL64R32 besitzt z. B. insgesamt 64 Eingänge für jeden einzelnen Produkt-Term, nämlich 32 Eingangsleitungen und 32 Rückkoppeleingänge der Register. Jedem Produkt-Term können daher 21 NAND-Gatter mit vier Eingängen zugeordnet werden. Zusätzlich wird das ODER-Gatter aus drei NAND-Gattern gebildet, und das Register zählt für sechs Gatterfunktionen. Mit dem zusätzlichen Ausgangstreiber ergibt das 178 Gatter für einen SMAC im PAL64R32. Die insgesamt 32 SMACs im Bauelement bedeuten also eine Komplexität für das PAL64R32 von 5696 Gatterfunktionen.

Bild 2 gibt eine Übersicht der äquivalenten Gatterkomplexität typischer PAL-Bausteine.

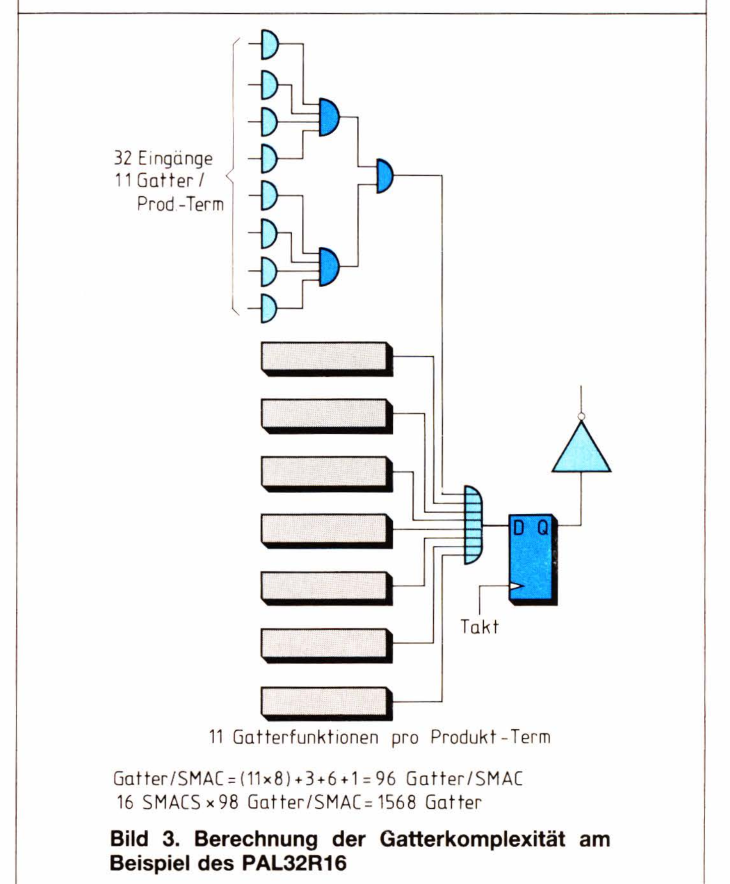


Bild 3. Berechnung der Gatterkomplexität am Beispiel des PAL32R16

Willibald Voldan

Der Simulator zum PALASM-II

Die Simulation ist ein besonders wichtiger Teil beim Entwickeln von digitalen Schaltungen mit Hilfe von Array-Anordnungen und programmierbarer Logik. Nachdem der Anwender seine Logikschaltung in Form Boole'scher Algebra definiert hat, ist es notwendig, zu verifizieren, daß diese Gleichungen die gewünschte Logikfunktion auch wirklich erfüllen. PALASM-II beinhaltet hierfür einen „Zustandsabhängigen Simulator", der die verschiedenen PAL-Konfigurationen von Monolithic Memories unterstützt. Das Programm wurde konzipiert, um intern erzeugte Zustände, die von asynchronen oder synchronen Rückkopplungen herrühren, besonders realistisch simulieren zu können. Schwingzustände werden dabei entdeckt und dem Anwender von der Software mitgeteilt. Ebenso erzeugen Unterschiede in den tatsächlichen und vom Anwender erwarteten Ausgangspegeln Fehlermeldungen. Der Simulator setzt dann von diesem Punkt seine Tätigkeit mit dem von ihm erwarteten Signalpegel fort.

Die Befehlsstruktur des Simulators im PALASM-II ist sehr einfach, aber auch höchst effizient ausgelegt. Die wesentlichen Unterschiede zu dem Simulationsvorgang im PALASM-I mit Hilfe der „Function Table" sind:

- Alle Befehle bestehen aus „englischen Wörtern", im Gegensatz zu den Pegelangaben (C,H,L,X,Z) des PALASM-I. Dies ermöglicht eine besonders strukturierte und übersichtliche Funktionsbeschreibung.
- Mit der Einführung von Schleifen wird besonders viel Zeit eingespart. Wurde in der Funktionstabelle z. B. ein 8-Bit-Zähler definiert, mußten, um alle Zustände durchzuspielen, 256 Vektoren mit jeweils acht Zuständen eingetippt werden. Das setzt der Simulation des PALASM-I Grenzen. Ein PAL64R32 besitzt z. B. 64 Eingänge und 32 Registerausgänge. Das ergibt eine komplette Funktionstabelle mit 32×2^{32} Testvektoren. Mit Funktionsschleifen können beliebig lange Sequenzen mit nur wenigen Kommandozeilen festgelegt werden.
- In der Funktionsprüfung ist es möglich, auf Grund von bestimmten Zuständen Verzweigungen in den Testvektoren auszuführen.
- Pegel, die sich von einem Vektor zum nächsten nicht verändern, müssen nicht in jedem Vektor neu definiert werden.
- Die Struktur des PALASM-II-Simulators erlaubt die Funktionsprüfung auch asynchroner Zustände und ist darüber hinaus auch in der Lage, Schwingzustände zu entdecken.

Das Resultat der Simulation sind Testvektoren, die in einer horizontalen Schreibweise ähnlich einem Pulsdiagramm aufgezeichnet sind. Jeder „Setz"- und „Takt"-Vorgang wird mit einem „g" bzw. „c" gekennzeichnet. Das hilft dem Anwender, die Simulation noch übersichtlicher und durchschaubarer zu halten.

Die wichtigsten Simulations-Befehle des PALASM-II:

Befehl SETF

SETF <Signal-Liste>

Beispiel: SETF A /OE B /RESET /D0 D1 D2

Dieser Befehl setzt die Signale auf logisch „HIGH", ist dem Signalnamen ein „/" vorangesetzt, wird der entsprechende Pegel auf „LOW" gesetzt. Im obigen Beispiel sind A, B, D1 und D2 auf „HIGH", sowie OE, RESET und D0 auf „LOW" gesetzt.

Ist das Signal einmal auf einen Pegel festgelegt, muß es nur neu gesetzt werden, wenn ein Pegelwechsel für diesen Pin durchgeführt wird. Der Simulator erinnert sich nämlich an den jeweils zuletzt definierten Zustand für ein Signal. Zu Beginn der Simulation sind alle Ein- und Ausgänge mit X („Don't Care") angenommen.

Jedesmal, wenn ein SETF-Befehl ausgeführt ist, wird ein Vektor erzeugt und alle vorhandenen Gleichungen damit verifiziert. Intern erzeugte Zustände werden ebenfalls erkannt und getestet. Tritt während der Vektor-Generierung ein Schwingzustand im PAL-Baustein auf, so werden solange weitere Testvektoren erzeugt, bis das System stabilisiert ist.

Beispiel:

```
CHIP BASIC_GATES PAL12P6

C D F G M N P Q I GND J K L R S H E B A VCC

EQUATIONS

B = /A                      ;inverter

E = C*D                     ;und-gatter

H = F+G                     ;oder-gatter

L = /J+/K                   ;nand-gatter

S = /M*/N                   ;nor-gatter

R = P*/Q + /P*Q             ;exclusive-or-gatter

SIMULATION

TRACE_ON A B C D E F G H J K L M N S P Q R

SETF /A /C /D /F /G /J /K /M /N /P /Q
SETF  A     D     G     K     N     Q
SETF     C /D  F /G  J /K  M /N  P /Q
SETF     C  D  F  G  J  K  M  N  P  Q
```

Befehl CLOCKF

CLOCKF <Signal-Liste>

Beispiel: CLOCKF CLK1 CLK2

Dem Befehl CLOCKF ist eine Signal-Liste zugeordnet, an welche Anschlüsse ein Taktsignal angelegt werden soll. Jeder CLOCKF-Befehl entspricht einer Signalform „LOW-HIGH-LOW“. Bei diesem Befehl werden daher zwei Vektoren erzeugt und während der positiven Taktflanke die neuen Zustände in die Register übernommen.

Bei jedem CLOCKF-Befehl werden intern erzeugte Zustände und asynchrone Vorgänge berücksichtigt. Bei Bedarf erzeugt der Simulator automatisch weitere Vektoren, bis das System zur Ruhe kommt.

Befehl CHECK

CHECK <Signal-Liste>

Beispiel: CHECK Q0 /Q1 /Q2

Diese Einrichtung ermöglicht es dem Anwender, die jeweiligen Pegel der Ausgangssignale zu jedem gewünschten Zeitpunkt zu überwachen.

In obigem Beispiel wird der Ausgang Q0 auf „HIGH“, Q1 und Q2 dagegen auf „LOW“ überprüft. Wie in der SETF-Anweisung werden Signale grundsätzlich auf „HIGH“, solche, denen ein „/“ vorsteht, auf „LOW“ überprüft.

Während der Ausführung eines CHECK-Kommandos vergleicht der Simulator die aktuellen Zustände der Ausgänge mit denen, die in dem CHECK-Befehl definiert sind. Bei einer Diskrepanz zwischen den beiden Signalen erfolgt eine Fehlermeldung, das Programm fährt von diesem Punkt an mit der Simulation mit den errechneten Signalpegeln fort.

Mit diesem Befehl lassen sich an wesentlichen Punkten der Simulation die Ausgänge überprüfen, um das Gesamt-System zu editieren.

Befehl TRACE-ON

TRACE-ON <Signal-Liste>

Beispiel: TRACE-ON /OE SET RESET D0 D1 D2 D3 /Q0 /Q1 /Q2

Dieser Befehl enthält die Liste der Signale, die während der Simulation stimuliert bzw. überprüft werden sollen. Im Ausgabe-Format sind diese Signale in der selben Anordnung wie im TRACE-ON-Befehl aufgelistet.

Die Signal-Liste ist solange gültig, bis ein TRACE-OFF-Befehl sie aufhebt oder die Simulation zu Ende ist. Nach dem TRACE-OFF-Kommando können dann wieder neue Signale mit dem TRACE-ON-Befehl eingeschaltet werden.

Diese Einrichtung ist dem Anwender behilflich, die Signale in einer von ihm gewünschten Anordnung aufzulisten, um damit die Übersichtlichkeit zu erhöhen.

Befehl TRACE-OFF

TRACE-OFF

Dieser Befehl schaltet alle Signale des letzten TRACE-ON-Kommandos ab. Nach diesem Befehl werden keine Ergebnisse der Simulation mehr erzeugt, bis der nächste TRACE-ON-Befehl ausgeführt ist. Alle Zustände zwischen dem TRACE-OFF- und dem darauffolgenden TRACE-ON-Befehl werden nicht angezeigt.

Diese Einrichtung hilft, die Resultate in verschiedene Zeitbereiche aufzuteilen, die für bestimmte Editiervorgänge wichtig sind.

FOR-Schleife

```
FOR I:= <untere Grenze> TO <obere Grenze>
  BEGIN
    <Befehl 1>
    <Befehl 2>
    .
    .
    .
  END
```

Beispiel:

```
FOR J := 3 TO 8
 BEGIN
  SETF A /B
  CLOCKF CLK
  IF J=1 THEN
    BEGIN
     CHECK Q0
    END
  ELSE
    BEGIN
     CHECK /Q0
    END
 END
```

Die FOR-Schleife bewirkt eine wiederholte Ausführung eines Befehls. Mit dieser Anweisung können viele Vektoren durch einfaches Erhöhen der Grenzen der FOR-Schleife erzeugt werden.

Die <untere Grenze> sollte kleiner oder gleich der oberen Grenze sein. Alle Grenzwerte müssen größer oder gleich 0 sein, negative Werte sind nicht zugelassen.

<u>IF THEN ELSE</u>

```
IF <Bedingung> THEN     oder    IF <Bedingung> THEN
 BEGIN                           BEGIN
  <S1>                            <S1>
  <S2>                            <S2>
 END                             END
ELSE
 BEGIN
  <S3>
  <S4>
END
```

Dieser Befehl kann in zwei Variationen auftreten. In der ersten Struktur tritt eine „ELSE"-Verzweigung auf, nicht aber in der zweiten, rechten Struktur. Ist die <Bedingung> erfüllt, wird die THEN-Sequenz ausgeführt, ist die <Bedingung> nicht erfüllt, wird die ELSE-Sequenz simuliert. Ist dagegen keine ELSE-Zeile vorhanden, wird automatisch der nächste Befehl ausgeführt.

Die <Bedingung> kann jede mathematische Gleichung darstellen, z. B.

=, >, <, >=, <>

Beispiel: IF (I<2) THEN

Die <Bedingung> kann auch aus Boole'schen Ausdrükken bestehen, z. B.

Beispiel: IF (DRDY * /CLR) THEN

Im ersten Beispiel wird die Bedingung auf I kleiner als 2 überprüft, im zweiten Beispiel ist der Ausdruck (DRDY * /CLR) ausgeführt; ist dieser erfüllt, so ist die Bedingung wahr.

Alle Signale werden zu Beginn der Simulation mit X („Don't Care") angenommen. Alle Steuer-Signale sollten daher zu Beginn auf ihren „Default"-Wert gesetzt werden. Wird zum Beispiel das „Enable"-Signal bei einigen PALs nicht gesetzt, treten während der Simulation Fehlermeldungen auf, da sich die Ausgänge im Three-State-Zustand befinden. Heißt der „Enable"-Anschluß z. B. /OE, dann gibt der Befehl „SETF OE" den Ausgang frei und „SETF /OE" bewirkt, daß der Ausgang im Three-State-Zustand ist.

Diese Bedingungen gelten auch für die „Preload"-Anschlüsse einiger PAL-Bausteine. Um die Register eines PAL-Bausteins vorzuladen, muß der „Preload"-Eingang auf „HIGH" gesetzt und die Ausgänge mit den gewünschten Pegeln versehen werden, zum Beispiel:

```
CHIP PAL20RA10
/PL /A B  NC NC NC NC NC NC NC NC GND
/OE Q0 Q1 Q2 NC NC NC NC NC NC NC VCC

EQUATIONS
.......

SIMULATION

SETF PL /OE Q0 /Q1 Q2   ;die Register Q0,Q1 und Q2
                        ;werden mit HLH vorgeladen.

SETF /PL OE             ;das PRELOAD Signal ist desaktiviert
                        ;und die Ausgänge werden freigegeben.

SETF /A B               ;Signal A wird auf "LOW"
                        ;und Signal B auf "HIGH" gesetzt.
```

Ist die Polarität von PL und OE „aktiv high" in der Pin-Liste, sollte ein korrespondierender Wechsel in der Polarität des SETF-Befehls vorkommen. „PRELOAD" und „OUTPUT ENABLE" sind festgelegte Anschlüsse eines PALs, die anders als herkömmliche Ein- und Ausgänge eine Inverter-Funktion durchlaufen.

Das PAL20RA10 besitzt eine komplett asynchrone Struktur. Jedes seiner zehn Ausgangsregister besitzt einen eigenen programmierbaren Takt-, Setz-, Rücksetz- und Three-State-Eingang. Für diese Struktur sind daher fünf voneinander unabhängige Gleichungen zur Beschreibung nötig:

```
Q0 := A * B
Q0.CLKF = CLK
Q0.RSTF = RESET
Q0.SETF = SET
Q0.TRS  = ENABLE

SETF SET /RESET         ;das Register Q0 wird auf HIGH gesetzt
                        ;und der Ausgangspin ist daher LOW.
SETF RESET /SET         ;das Register Q0 wird auf LOW gesetzt
                        ;und der Ausgangspin ist daher HIGH.
```

Von der Boole'schen Gleichung zur Simulation

Ein kompletter Vor-/Rück-Zähler mit 8 Bit läßt sich in einem PAL20X8 unterbringen. Er genügt den in *Listing Nr. 1* (folgende Seite) gezeigten Boole'schen Gleichungen. Aus dieser formalen Beschreibung erzeugt der PALASM-I die in *Listing Nr. 2* dargestellte Funktionstabelle zur Simulation.

Gegenüber dem PALASM-I bietet die Version II die Möglichkeit, Makro-Definitionen und -Gleichungen zu verwenden. Die Programmierarbeit wird noch weiter vereinfacht durch Stringsubstitution. *Listing Nr. 3* entspricht voll dem Listing Nr. 1; hieraus entsteht die in *Listing Nr. 4* gezeigte Funktionsbeschreibung zur Simulation, die sicher „lesbarer" ist als das Listing Nr. 2.

①
```
CHIP 8_Bit_Counter PAL20X8
CLK UP  D0 D1 D2 D3 D4 D5 D6 D7 LD  GND
/OC SET Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 CIN VCC

EQUATIONS

/Q0 :=  /SET* LD*/D0                               ;LOAD D0
      + /SET*/LD*/Q0                               ;HOLD Q0
    :+: /SET*/LD* CIN* UP                          ;INCREMENT
      + /SET*/LD* CIN*/UP                          ;DECREMENT

/Q1 :=  /SET* LD*/D1                               ;LOAD D1
      + /SET*/LD*/Q1                               ;HOLD Q1
    :+: /SET*/LD* CIN* UP* Q0                      ;INCREMENT
      + /SET*/LD* CIN*/UP*/Q0                      ;DECREMENT

/Q2 :=  /SET* LD*/D2                               ;LOAD D2
      + /SET*/LD*/Q2                               ;HOLD Q2
    :+: /SET*/LD* CIN* UP* Q0* Q1                  ;INCREMENT
      + /SET*/LD* CIN*/UP*/Q0*/Q1                  ;DECREMENT

/Q3 :=  /SET* LD*/D3                               ;LOAD D3
      + /SET*/LD*/Q3                               ;HOLD Q3
    :+: /SET*/LD* CIN* UP* Q0* Q1* Q2              ;INCREMENT
      + /SET*/LD* CIN*/UP*/Q0*/Q1*/Q2              ;DECREMENT

/Q4 :=  /SET* LD*/D4                               ;LOAD D4
      + /SET*/LD*/Q4                               ;HOLD Q4
    :+: /SET*/LD* CIN* UP* Q0* Q1* Q2* Q3          ;INCREMENT
      + /SET*/LD* CIN*/UP*/Q0*/Q1*/Q2*/Q3          ;DECREMENT

/Q5 :=  /SET* LD*/D5                               ;LOAD D5
      + /SET*/LD*/Q5                               ;HOLD Q5
    :+: /SET*/LD* CIN* UP* Q0* Q1* Q2* Q3* Q4      ;INCREMENT
      + /SET*/LD* CIN*/UP*/Q0*/Q1*/Q2*/Q3*/Q4      ;DECREMENT

/Q6 :=  /SET* LD*/D6                               ;LOAD D6
      + /SET*/LD*/Q6                               ;HOLD Q6
    :+: /SET*/LD* CIN* UP* Q0* Q1* Q2* Q3* Q4* Q5  ;INCREMENT
      + /SET*/LD* CIN*/UP*/Q0*/Q1*/Q2*/Q3*/Q4*/Q5  ;DECREMENT

/Q7 :=  /SET* LD*/D7                               ;LOAD D7
      + /SET*/LD*/Q7                               ;HOLD Q7
    :+: /SET*/LD* CIN* UP* Q0* Q1* Q2* Q3* Q4* Q5* Q6   ;INCREMENT
      + /SET*/LD* CIN*/UP*/Q0*/Q1*/Q2*/Q3*/Q4*/Q5*/Q6   ;DECREMENT
```

②
```
FUNCTION TABLE
CLK SET LD CIN UP /OC
D0 D1 D2 D3 D4 D5 D6 D7 Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7

;C S   C   /   - DATA IN-   DATA OUT
;L E L I U O   DDDDDDDD     QQQQQQQQ
;K T D N P C   01234567     01234567    COMMENT
-------------------------------------------------------------
 C H X X X L   XXXXXXXX     HHHHHHHH    SET (SET=H)
 C L L H H L   XXXXXXXX     LLLLLLLL    CLEAR, START TO COUNT UP
 C L L H H L   XXXXXXXX     HLLLLLLL    COUNT UP {01}
 C L L H H L   XXXXXXXX     LHLLLLLL    COUNT UP {02}
 C L L H H L   XXXXXXXX     HHLLLLLL    COUNT UP {03}
 C L L H H L   XXXXXXXX     LLHLLLLL    COUNT UP {04}
 C L L H H L   XXXXXXXX     HLHLLLLL    COUNT UP {05}
 C L L H H L   XXXXXXXX     LHHLLLLL    COUNT UP {06}
 C L L H H L   XXXXXXXX     HHHLLLLL    COUNT UP {07}
 C L L H H L   XXXXXXXX     LLLHLLLL    COUNT UP {08}
 C L L L H L   XXXXXXXX     LLLHLLLL    OUTPUTS HOLD THEIR VALUES
 C L L L H L   XXXXXXXX     LLLHLLLL    OUTPUTS HOLD THEIR VALUES
 C L L H L L   XXXXXXXX     HHHLLLLL    COUNT DOWN {07}
 C L L H L L   XXXXXXXX     LHHLLLLL    COUNT DOWN {06}
 C L L H L L   XXXXXXXX     HLHLLLLL    COUNT DOWN {05}
 C L H H L L   LHLHLHLH     LHLHLHLH    LOAD DATA {AA}
 C L H H L L   LLHHLLHH     LLHHLLHH    LOAD DATA {CC}
 C L L H H L   LLHHLLHH     HLHHLLHH    COUNT UP {CD}
 C L L H H L   LLHHLLHH     LHHHLLHH    COUNT UP {CE}
 C L L H H L   LLHHLLHH     HHHHLLHH    COUNT UP {CF}
 C L L H H L   LLHHLLHH     LLLLHLHH    COUNT UP {D0}
 C L L H L L   LLHHLLHH     HHHHLLHH    COUNT DOWN {CF}
 C L L H L L   LLHHLLHH     LHHHLLHH    COUNT DOWN {CE}
 C L L H L L   LLHHLLHH     HLHHLLHH    COUNT DOWN {CD}
 C L L H L L   LLHHLLHH     LLHHLLHH    COUNT DOWN {CC}
 C L L H L L   LLHHLLHH     HHLHLLHH    COUNT DOWN {CB}
 C L L H L L   LLHHLLHH     LHLHLLHH    COUNT DOWN {CA}
 C L L H L H   LLHHLLHH     ZZZZZZZZ    SET OUTPUTS HI-Z
-------------------------------------------------------------
```

③
```
CHIP 8_BIT_COUNTER

CLK UP  D(0..7) LD  GND
/OC SET Q(7..0) CIN VCC

STRING COUNT  "/SET*/LD*CIN"                      ;equate Macro

EQUATIONS

/Q(I=0..7) :=  /SET* LD*/D(I)                     ;load D(I)
             + /SET*/LD*/Q(I)                     ;hold Q(I)
           :+:  Q(I-1)*..* Q(0)* UP*COUNT         ;increment
             +  /Q(I-1)*..*/Q(0)*/UP*COUNT        ;decrement

COUT        =   Q(7)*..*Q(0)*CIN                  ;carry out
```

④
```
SIMULATION

TRACE_ON CLK SET LD CIN UP /OC
         D0 D1 D2 D3 D4 D5 D6 D7          ;these inputs/outputs are
         Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7          ;traced on during simulation

SETF OC SET                               ;SET=H  OC=L
CLOCKF CLK                                ;preset outputs to high
CHECK Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0             ;check outputs for all HIGH

SETF /SET /LD CIN UP                      ;prepare control inputs
                                          ;for counting up
FOR I:= 1 TO 8 DO                         ;8 clock pulses
  BEGIN
    CLOCKF CLK                            ;count up {01} - {08}
      IF I=6 THEN
        BEGIN                             ;check outputs after
          CHECK /Q7 /Q6 /Q5 /Q4           ;6 clock pulses for {06}
                /Q3  Q2  Q1 /Q0
        END
  END
CHECK /Q7 /Q6 /Q5 /Q4                     ;check outputs after
       Q3 /Q2 /Q1 /Q0                     ;8 clock pulses for {08}

SETF /CIN                                 ;disable count
CLOCKF CLK                                ;outputs hold their values
CLOCKF CLK                                ;outputs hold their values
CHECK /Q7 /Q6 /Q5 /Q4                     ;check outputs for {08}
       Q3 /Q2 /Q1 /Q0

SETF /UP CIN                              ;prepare control inputs
                                          ;for counting down
FOR I:= 1 TO 3 DO                         ;3 clock pulses
  BEGIN
    CLOCKF CLK                            ;count down {08} - {05}
  END
CHECK /Q7 /Q6 /Q5 /Q4                     ;check outputs for {05}
      /Q3  Q2 /Q1  Q0

SETF LD /D0 D1 /D2 D3
        /D4 D5 /D6 D7                     ;prepare data to be loaded
CLOCKF CLK                                ;load data {AA}
CHECK  Q7 /Q6 Q5 /Q4                      ;check outputs for {AA}
       Q3 /Q2 Q1 /Q0

SETF    /D0 /D1 D2 D3
        /D4 /D5 D6 D7                     ;prepare data to be loaded
CLOCKF CLK                                ;load data {CC}
CHECK  Q7 Q6 /Q5 /Q4                      ;check outputs for {CC}
       Q3 Q2 /Q1 /Q0

SETF /LD UP                               ;count up
FOR J:= 1 TO 10 DO                        ;apply 10 clock pulses
  BEGIN
    CLOCKF CLK                            ;count up
    IF J=4 THEN                           ;condition jump
      BEGIN                               ;to change the function
      SETF /UP                            ;count down
      END
  END
SETF /OC                                  ;three-state outputs
```

Dr.-Ing. Günter Biehl, Dr.-Ing. Albrecht Ditzinger

LOGE – Programmierbare Logik problemlos entwerfen

PAL-, FPLA-, FPLS-, PROM- und weitere programmierbare Logikbausteine geben dem Entwickler digitaler Schaltungen die Möglichkeit, komplexe Aufgabenstellungen kompakt zu realisieren. Zur vollen Ausnutzung ihrer Möglichkeiten und ihrer Flexibilität ist allerdings ausreichende Rechnerunterstützung bei allen Entwurfsschritten, wie z. B. bei der booleschen Minimierung, erforderlich. Die bisher verfügbaren Programme beschränkten sich auf bausteinspezifische Aufgaben. Die mühsamen und fehleranfälligen Entwurfsschritte blieben dem Entwickler überlassen. Dieser Beitrag stellt das Programmsystem LOGE vor und beschreibt, wie es Entwurf und Optimierung von programmierbarer Logik vereinfacht und beschleunigt.

Die vielfältigen Vorteile programmierbarer Logikbausteine haben in jüngster Zeit zu immer weiterer Verbreitung dieser hochintegrierten Schaltungen geführt [1]. Von großem Interesse sind insbesondere solche Bausteine, die vom Anwender selbst programmiert werden können („field programmable"). PROM-, EPROM-, PAL-, FPLA- und FPLS-Bausteine gehören zu dieser Gruppe.

Ihre hohe Integrationsdichte ermöglicht es, durch einen einzigen derartigen Baustein mehrere MSI- und SSI-Chips zu ersetzen und damit Leiterplattenfläche einzusparen. Die durch die Programmierfähigkeit gewonnene Flexibilität erlaubt es dabei, auch mit einem kleinen Lagersortiment weitgefächerte Anwendungsgebiete abzudecken.

1 Problematik des Logikentwurfs

Den offensichtlichen Vorteilen dieser komplexen Bauelemente stehen die Schwierigkeiten ihrer Anwendung in der Entwurfspraxis gegenüber:

Auch bei hochintegrierten Bausteinen beeinflussen die strukturellen Grenzwerte den Entwurf entscheidend. *Tabelle 1* zeigt, welche Grenzwerte bei der Implementierung von Logikfunktionen mit den einzelnen Bausteintypen besonders kritisch sind. Logikentwürfe nach einem Ad-hoc-Ansatz führen oft dazu, daß die Funktion nicht in den Baustein paßt, weil mindestens einer der Grenzwerte von Tabelle 1 überschritten wird. Daran scheitert dann die Realisierung mit dem gewünschten Bauelement. Durch Optimierung des Logikentwurfs, z. B. durch boolesche Minimierung, könnte der Grenzwert oft eingehalten werden, jedoch sind entsprechende Optimierungsverfahren so rechenaufwendig, daß sie „von Hand" nur für kleine Problemstellungen durchgeführt werden können. Die Konsequenz daraus ist, daß beim bisher noch weitverbreiteten „Handentwurf" die prinzipiellen Möglichkeiten der Bausteine längst nicht ausgenutzt werden. Insbesondere werden PAL-Bausteine oft nur dazu genutzt, einige verstreute UND- bzw. ODER-Gatter einer Platine in einem Bauelement zu konzentrieren.

Eine weitere Schwierigkeit beim herkömmlichen Entwurfsstil stellt der Nachweis der Korrektheit der Lösung dar. Bei der großen Menge von Daten, die während der Umsetzung einer Logikaufgabenstellung in die Programmiermuster zu bewältigen ist, schleichen sich sehr leicht Fehler ein, die nachher oft nur in tagelangen Tests gefunden werden können.

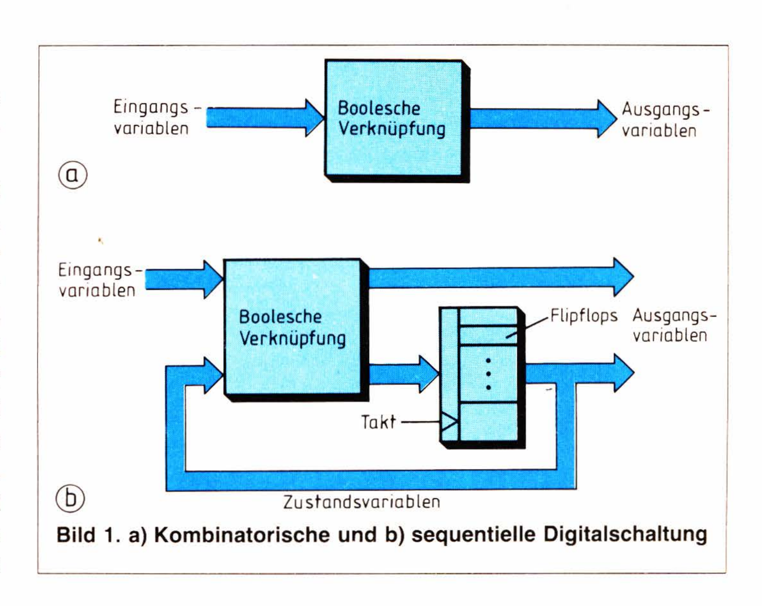


Bild 1. a) Kombinatorische und b) sequentielle Digitalschaltung

Aus diesen Gründen bemühen sich die Hersteller um Software zur Rechnerunterstützung des Logikentwurfs für ihre Bausteintypen. Für FPLA- und FPLS-Bausteine z. B. wird vom Hersteller das Softwarepaket AMAZE angeboten, das die Umsetzung logischer Gleichungen im Programmiermuster durchführt.

Für PAL-Bausteine stellt das Programm PALASM [2] einen ersten Ansatz von Rechnerunterstützung dar. Es führt die Umsetzung von logischen Gleichungen in die Programmiermuster für PALs automatisch durch und entlastet den Logikentwerfer damit von einem sonst sehr mühevollen und fehleranfälligen Schritt. Man muß sich jedoch vergegenwärtigen, daß die logischen Gleichungen, von denen PALASM ausgeht, bereits das *Ergebnis* des eigentlichen Logikentwurfs darstellen. Dieser muß also auch beim Einsatz von PALASM vom Entwickler selbst durchgeführt werden.

Wegen der dabei weiterhin bestehenden Fehlergefahr beinhaltet PALASM die Möglichkeit zur Logiksimulation: Das Sollverhalten der Logikschaltung ist dazu vom Entwerfer in einer Funktionstabelle festzuschreiben. PALASM überprüft die logischen Gleichungen auf Äquivalenz mit der Funktionstabelle.

2 Entwurf mit Hilfe des Programmsystems LOGE

Im Gegensatz dazu stellt das Programmsystem LOGE ein umfassendes Hilfsmittel dar, das auch den eigentlichen Logikentwurf mit Hilfe von optimierenden Syntheseverfahren automatisch durchführt. Da LOGE als Ausgangspunkt der Synthese die erwähnte Funktionsbeschreibung verwendet, ist die Übereinstimmung der Lösung mit der Funktionsbeschreibung von vornherein gewährleistet. Die Notwendigkeit der Logiksimulation entfällt, Neuentwürfe aufgrund von Logikfehlern sind nicht erforderlich, und die Entwicklungszeit verkürzt sich entsprechend.

Ergebnis von LOGE sind z. B. für den PAL-Entwurf die logischen Gleichungen als Eingabe für PALASM bzw. beim FPLA-/FPLS-Entwurf die Programmiermuster für die Bausteine.

Zur genaueren Vorstellung der Leistungsfähigkeit von LOGE müssen die beiden grundlegenden Klassen von Digitalschaltungen gemäß *Bild 1* unterschieden werden: Bei *kombinatorischen* Schaltungen hängt zu jedem Zeitpunkt der Wert der Ausgangsvariablen allein vom Wert der Eingangsvariablen ab. Im Gegensatz dazu haben *sequentielle* Schaltungen ein Gedächtnis, d. h. die Vorgeschichte beeinflußt den inneren Zustand der Schaltung und damit das Ausgangsverhalten. Dieser Zustand der Schaltung manifestiert sich im jeweiligen Inhalt der Speicherflipflops.

2.1 Kombinatorische Logik

2.1.1 Aufgabenbeschreibung durch die Funktionstabelle

Entwerfen bedeutet für kombinatorische Logik, wie für jede andere Technik auch, das Umsetzen der Aufgabenstellung in eine konkrete Realisierung. Deshalb muß zu Beginn des Entwurfs das Sollverhalten der Schaltung festgeschrieben werden. Das geschieht zweckmäßigerweise in Form der bereits erwähnten Funktionstabelle (auch Wahrheitstabelle), die unmittelbar als Eingabe für LOGE dient. Diese listet für alle Wertekombinationen der Eingangsvariablen die zugeordneten Werte der Ausgangsvariablen der Schaltung tabellarisch auf. *Tabelle 2* zeigt ein Beispiel.

Besonders kompakt ist diese Tabelle dadurch, daß in den Kombinationen der Eingangsseite die Werte „—" zugelassen sind. Das bedeutet, daß sowohl für den Wert 0 als auch für den Wert 1 der betreffenden Eingangsvariablen diese Zeile gelten soll. So steht Zeile 2 von Tabelle 2 für die vier Wertekombinationen 0001, 0011, 0101 und 0111. Der Wert „Rest" in Zeile 5 steht für alle in der Tabelle nicht angeführten Wertekombinationen.

Auf der Ausgangsseite der Tabelle bedeutet der Wert „–", daß die betreffende Ausgangsvariable in dieser Zeile einen beliebigen Wert (*don't care*) annehmen darf. Es ist sehr zweckmäßig, diesen Sachverhalt explizit anzugeben, weil LOGE diese Freiheitsgrade für die Optimierung der Schaltung ausnutzt.

2.1.2 Optimierung des Entwurfs

Die Optimierung hat hier zum Ziel, die Möglichkeiten des programmierbaren Bausteins möglichst gut zu nutzen, d. h. dafür zu sorgen, daß die Aufgabenstellung in den Chip „hineinpaßt". Für die Bausteintypen PAL und FPLA, die intern eine zweistufige UND/ODER- bzw. ODER/UND-Struktur enthalten, kommt es darauf an, die Aufgabenstellung mit möglichst wenigen Produktter-

Tabelle 1. Grenzwerte von programmierbaren Logikbausteinen

Grenzwert der Bausteinstruktur	kritisch für
Produkttermzahl	PAL (8–16), FPLA (48), FPLS (48)
Anzahl der Produktterme je Ausgangsfunktion	PAL (8–16)
Anzahl der integrierten Flipflops und Rückführungen	PAL-R-Typen (4–10) FPLS (6)
Anzahl der Ein-/Ausgabepins	alle Typen

Tabelle 2. Beispiel für eine Funktionstabelle

	Eingangs-variablen				Ausgangs-variablen				
1	0	1	0	0	–	1	1	0	1
2	0	–	–	1	0	1	–	0	–
3	1	1	–	0	1	1	–	0	0
4	–	0	0	0	0	0	1	1	0
5	Rest				–	–	0	–	–

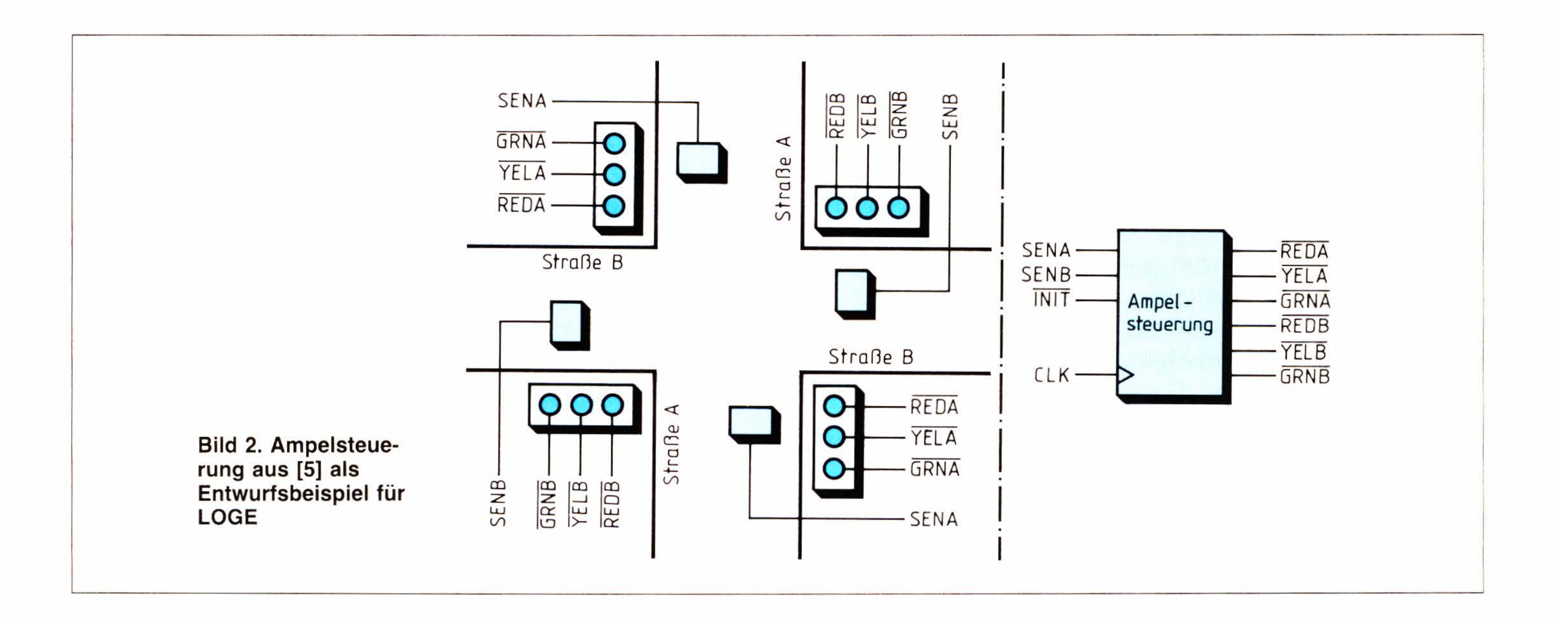


Bild 2. Ampelsteuerung aus [5] als Entwurfsbeispiel für LOGE

men realisieren zu können. Für diese boolesche Minimierung sind in LOGE Algorithmen implementiert, die im Falle des PAL-Entwurfs die Produkttermzahl jeder einzelnen Ausgangsvariablen minimieren und im Falle des FPLA-Entwurfs die Produkttermzahl insgesamt (sogenannte Bündelminimierung).

Die in LOGE implementierten Algorithmen wurden völlig neu entwickelt und speziell auf die Belange des rechnergestützten Logikentwurfs zugeschnitten. Sie übertreffen dadurch das bisher am weitesten verbreitete Verfahren nach *Quine* und *McCluskey* in Speicher- und Rechenzeiteffizienz um Größenordnungen. Dadurch können auch umfangreiche Aufgabenstellungen mit kurzen Rechenzeiten, im Mittel im Sekundenbereich, minimiert werden. Die Grenzen von LOGE liegen (für Installationen auf 32-Bit-Rechnern) bei je 64 Ein- und Ausgangsvariablen. Wenn man sich vergegenwärtigt, daß Funktionstabellen mit etwa acht Ein- und Ausgangsvariablen von Hand bereits nicht mehr optimiert werden können, wird die Leistungsfähigkeit von LOGE unmittelbar einsichtig.

Darüber hinaus erkennt LOGE auch, wenn Ausgangsvariablen einander gleich oder zueinander invers sind. In diesen Fällen brauchen nämlich keine wertvollen Produktterme verschwendet zu werden, da die betreffenden Ausgangsvariablen durch einfache Verbindungen oder Invertierer realisiert werden können. Bei größeren Aufgabenstellungen ist selbst diese offensichtliche Optimierungsmöglichkeit, obwohl sehr einfach, wegen der großen Datenmengen vom Entwerfer nicht auf Anhieb zu erkennen. Insbesondere können die Möglichkeiten, die aus den Don't-care-Einträgen in der Funktionstabelle resultieren, nur durch ein Rechnerprogramm optimal genutzt werden.

2.2 Sequentielle Logik

Eine ganz erhebliche Erweiterung der Möglichkeiten der programmierbaren Logikelemente stellen die mit getakteten Flipflops und Rückführungen auf dem Chip ausgestatteten Bausteine (PALs der R-Serie, FPLS) dar. Mit diesen Bausteinen hat der Entwickler die Möglichkeit, sequentielle Schaltungen in einen einzigen Chip zu integrieren und so spezielle Controller-Chips für seine Aufgabe zu entwerfen. Allerdings stellt sich auch hier das Problem, diese Schaltung so zu entwickeln, daß sie „in den Chip hineinpaßt“.

Die bisher oft geübte Entwurfspraxis, die einzelnen Zustände der Schaltung in Einzelflipflops festzuhalten, erscheint wegen der geringen Anzahl der in den Bausteinen verfügbaren Flipflops wenig sinnvoll; beispielsweise könnte in einem derzeit verfügbaren FPLS mit sechs Flipflops nur eine Schaltung mit sechs Zuständen untergebracht werden, wodurch die Möglichkeiten dieses Bausteins natürlich in keiner Weise ausgenutzt würden. Sinnvoll erscheint daher die auch in [5, 6] vorgeschlagene Vorgehensweise, die Schaltung als sogenannte „State Machine“ aufzufassen und die Funktion durch ein grafisches Hilfsmittel, etwa den bekannten Zustandsgraphen, anzugeben. Die Zustände dieses Graphen können dann kompakt codiert in den Flipflops gespeichert werden. Auf diese Weise wäre in dem zuvor erwähnten FPLS eine Schaltung mit $2^6 = 64$ Zuständen unterzubringen, wodurch die Möglichkeiten des Bausteins in der Regel ausgeschöpft werden.

Die zur Umwandlung des Zustandsgraphen in die Schaltung nötigen Entwicklungsschritte sind seit langem bekannt und lassen sich mit den Stichworten
- Zustandsreduktion,
- Aufstellen der Flipflop-Ansteuerfunktionen,
- Optimierung der Ansteuer- und Ausgabefunktionen

umschreiben. Insbesondere diesem letzten Punkt kommt für PAL und FPLS entscheidende Bedeutung zu, da die Güte der Optimierung darüber entscheidet, ob ein Problem mit den im Baustein vorhandenen Termen realisierbar ist oder nicht. Gerade für die Optimierung aber stehen für den Handentwurf nur unzureichende Hilfsmittel zur Verfügung, so etwa KV-Diagramme für die

boolesche Minimierung. Deshalb wird der komplette Entwurfsprozeß sehr unhandlich, unübersichtlich und fehleranfällig.

An dieser Stelle kann das System LOGE den Entwickler wirksam unterstützen, da es alle angeführten Entwurfsschritte automatisch ausführt. Auch die Größenordnung der bearbeitbaren Probleme stellt kein Hindernis dar, da Aufgaben mit bis zu 1024 Zuständen bearbeitet werden können. Die Zahl der bearbeitbaren Ansteuer- und Ausgabefunktionen beträgt 64, wobei jede Funktion von maximal 64 Variablen abhängen kann. Mit dieser Leistungsfähigkeit lassen sich alle derzeit auf dem Markt befindlichen Bausteine problemlos erfassen.

Bild 3. Aufgabenbeschreibung für sequentielle Schaltungen durch ein Ablaufdiagramm

3 Entwurfsbeispiel

Zur Demonstration des Entwurfs mit Hilfe des Systems LOGE soll ein Beispiel dienen, für das in [5] ein vollständiger Handentwurf beschrieben ist. Das sehr einfache Beispiel, eine Verkehrsampelsteuerung, wurde durch Einführung der hierzulande üblichen Rot/Gelb-Phase und eines Einschaltzustandes, in dem alle Ampeln gelb leuchten, noch etwas komplizierter gestaltet. Es geht von einer Ampel an der Kreuzung von Straße A mit Straße B gemäß *Bild 2* aus. In beiden Straßen liegen Induktionsschleifen, die die Signale SENA (sense A) und SENB liefern. Zusätzlich ist noch ein Eingangssignal INIT vorhanden, das die Schaltung in den Einschaltzustand versetzt. Die Ausgangssignale tragen die selbsterklärenden Namen REDA, YELA, GRNA, REDB, YELB und GRNB.

Mit dieser Beschreibung läßt sich bereits die grafische Darstellung der Schaltungsfunktion erstellen. Für das System LOGE wird an Stelle des Zustandsgraphen, der für komplexe Aufgaben sehr unübersichtlich werden kann, eine etwas veränderte Darstellung, das sogenannte Ablaufdiagramm, verwendet. *Bild 3* zeigt das Ablaufdiagramm für das Ampel-Beispiel. Man erkennt die Zustände an rechteckigen Symbolen, an deren oberem Ende der Takteinfluß durch einen schwarzen Balken dargestellt ist. Darunter wird jeweils die Soll-Ausgabe in diesem Zustand notiert (Beispiel: in Zustand 13 YELA = 1 und YELB = 1: ein nachgestellter Schrägstrich bezeichnet das invertierte Signal). Es folgt die Verzweigung, durch die abhängig von den angegebenen Variablen einer von mehreren Wegen eingeschlagen wird (Beispiel: In Zustand 13 wird die Variable INIT abgefragt, hat sie den Wert 1, wird mit dem nächsten Takt nach Zustand 1 verzweigt, bei 0 wird Zustand 13 beibehalten). Mit dieser einfachen Symbolik läßt sich die Funktion auch eines komplexen Controllers eindeutig beschreiben.

Mit Vorliegen des Ablaufdiagramms ist die eigentliche Entwurfsarbeit beendet, und das Problem kann zur Optimierung in den Rechner eingegeben werden. Da in der Regel kein grafisches Terminal zur Verfügung steht, geschieht diese Eingabe über die sogenannte Ablauftabelle, die sich ganz einfach aus dem Diagramm ableiten läßt.

Bild 4 zeigt den vollständigen LOGE-Datensatz für das Ampel-Beispiel, in dessen Mittelpunkt die Ablauftabelle steht. Die Kodierung der einzelnen Zustände, die hinter der Ablauftabelle angeführt ist, wurde in Anbetracht der im PAL vorhandenen Ausgangsinverter so gewählt, daß an den Bausteinausgängen direkt die Ausgangsfunktionen entstehen.

Bild 5 zeigt das von LOGE in tabellarischer Form erzeugte Minimierungsergebnis in Form von booleschen

```
;
; Ampelsteuerung aus MMI PAL-Handbook,
; erweitert um Rot-Gelb Phase.
; Mit LOGE problemlos in einem PAL 16R8
; zu realisieren.
;
; Signal ein bei 0 am PAL pin, Inverter im PAL
; werden von LOGE automatisch beruecksichtigt
;
; Zeitaufwand fuer die gesamte Entwicklung
; incl. Dateneingabe:  etwa eine Stunde
;
; LOGE Rechenzeit auf IBM-XT : 103 sec
;
*PAL
 TYPE=PAL16R8
*IDENTIFICATION
 TRL CONTROLLER
*DECLARATIONS
 X-VARIABLES = 3
 Z-VARIABLES = 8
 CPU-TIME = 100
*X-NAMES
 SENA = 1, SENB = 2, INIT = 3 ;
*Z-NAMES
 REDA = 1, YELA = 2, GRNA = 3,
 REDB = 4, YELB = 5, GRNB = 6,
 Q7   = 7, Q8   = 8 ;
*Z-VALUES
 S1  = 110 011 11 ; Gruen    auf A, Rot      auf B
 S2  = 110 011 10 ; Gruen    auf A, Rot      auf B
 S3  = 110 011 00 ; Gruen    auf A, Rot      auf B
 S4  = 110 011 01 ; Gruen    auf A, Rot      auf B
 S5  = 101 011 -- ; Gelb     auf A, Rot      auf B
 S6  = 011 001 -- ; Rot      auf A, Rot/Gelb auf B
 S7  = 011 110 11 ; Rot      auf A, Gruen    auf B
 S8  = 011 110 10 ; Rot      auf A, Gruen    auf B
 S9  = 011 110 00 ; Rot      auf A, Gruen    auf B
 S10= 011 110 01 ; Rot      auf A, Gruen    auf B
 S11= 011 101 -- ; Rot      auf A, Gelb     auf B
 S12= 001 011 -- ; Rot/Gelb auf A, Rot      auf B
 S13= 101 101 -- ; Gelb     auf A, Gelb     auf B
*FLOW TABLE
S  1,  X --0  ,  F 13  ; INIT
S  1,  X 001  ,  F  2  ; Kein Fahrzeug, starte Zeit A
S  1,  X 111  ,  F  2  ; Fahrzeuge auf A und B, starte Zeit A
S  1,  X 101  ,  F  1  ; Fahrzeuge nur auf A, A bleibt gruen
S  1,  X 011  ,  F  5  ; A frei, Fahrzeug auf B, A wird gelb
;
S  2,  X --0  ,  F 13  ; INIT
S  2,  X --1  ,  F  3  ; Wartezeit A
;
S  3,  X --0  ,  F 13  ; INIT
S  3,  X --1  ,  F  4  ; Wartezeit A
;
S  4,  X --0  ,  F 13  ; INIT
S  4,  X --1  ,  F  5  ; Wartezeit A
;
S  5,  X --0  ,  F 13  ; INIT
S  5,  X --1  ,  F  6  ; Wartezeit A
;
S  6,  X --0  ,  F 13  ; INIT
S  6,  X --1  ,  F  7  ; Rot/Gelb auf B
;
S  7,  X --0  ,  F 13  ; INIT
S  7,  X 011  ,  F  7  ; Fahrzeug nur auf B, B bleibt gruen
S  7,  X 101  ,  F 11  ; Fahrzeug nur auf A, Gelb auf B
S  7,  X 001  ,  F  8  ; Kein Fahrzeug, Starte Zeit B
S  7,  X 111  ,  F  8  ; Fahrzeuge auf A und B, Starte Zeit B
;
S  8,  X --0  ,  F 13  ; INIT
S  8,  X --1  ,  F  9  ; Wartezeit B
;
S  9,  X --0  ,  F 13  ; INIT
S  9,  X --1  ,  F 10  ; Wartezeit B
;
S 10,  X --0  ,  F 13  ; INIT
S 10,  X --1  ,  F 11  ; Wartezeit B
;
S 11,  X --0  ,  F 13  ; INIT
S 11,  X --1  ,  F 12  ; Wartezeit B
;
S 12,  X --0  ,  F 13  ; INIT
S 12,  X --1  ,  F  1  ; Rot/Gelb auf A
;
S 13,  X --0  ,  F 13  ; Warte bis INIT verschwindet
S 13,  X --1  ,  F  1  ; Starte normalen Ablauf
*STATE-ASSIGNMENT
 Z-VARIABLES
*PINS
 SYSCLK =  1,
 SENA   =  2, SENB   =  3, INIT   =  4,
 Q8     = 12, Q7     = 13,
 GRNB   = 14, YELB   = 15, REDB   = 16,
 GRNA   = 17, YELA   = 18, REDA   = 19;
*END
```

Bild 4. Diese tabellarische LOGE-Eingabe entspricht der Aufgabenbeschreibung, wie sie in Bild 3 dargestellt ist

Gleichungen, so wie sie zur Eingabe für PALASM benötigt werden. Die Ausgabe von PALASM kann dann direkt zur PAL-Programmierung herangezogen werden, so daß die vollständige Entwurfskette von der grafischen Funktionsbeschreibung bis zum fertigen Baustein geschlossen ist.

Aus Bild 5 kann auch entnommen werden, daß die automatisch entworfene Schaltung problemlos durch ein PAL des Typs 16R8 realisiert werden kann. Ein Vergleich mit dem in [5] beschriebenen Handentwurf zeigt, welche Vielzahl an Tabellen, Diagrammen und fehleranfälligen Umsetzungen, bei zumindest gleichwertigem Ergebnis, durch den Einsatz des Systems LOGE entfallen kann.

4 Allgemeines Entwurfskonzept

Die hier vorgestellten Module des Programmsystems LOGE stellen nur einen Ausschnitt des Gesamtsystems dar, das noch sehr viel weitergehende Strukturen wie mikroprogrammierte Steuerungen oder Mikrorechner unterstützt. Da die ausführliche Beschreibung aller Module hier zu weit führen würde, soll nur das dem

```
PAL16R8
TRL CONTROLLER
85/06/11   8:48:08
COMPANY: ISDATA GMBH, 7500 KARLSRUHE 1, HAID-UND-NEU-STR. 7
SYSCLK SENA   SENB   INIT   NC     NC     NC     NC     NC     GND
OE     Q8     Q7     GRNB   YELB   REDB   GRNA   YELA   REDA   VCC

          /REDA   :=  REDA   *  GRNA   *  YELB   *  INIT
                   + /REDA   *  YELA   *  INIT

          /YELA   := /INIT
                   +  YELA   *  YELB   *  GRNB   * /Q7    *  Q8
                   +  YELA   *  REDB   *  GRNB
                   +  REDA   *  YELA   *  Q8     * /SENA  *  SENB

          /GRNA   :=  REDA   *  YELA   *  Q7     * /SENB  *  INIT
                   + /REDA   * /YELA   *  INIT
                   +  REDA   *  YELA   * /Q8     *  INIT
                   +  REDA   * /YELB   *  INIT
                   +  REDA   *  YELA   *  Q7     *  SENA  *  INIT

          /REDB   :=  REDB   *  GRNB   *  INIT
                   + /REDB   *  YELB   *  INIT

          /YELB   := /INIT
                   +  REDB   *  YELB   * /Q7     *  Q8
                   + /REDA   *  YELA   *  YELB   *  Q8    *  SENA
                             * /SENB
                   +  REDA   *  GRNA   *  YELB

          /GRNB   :=  YELA   *  GRNA   * /REDB   *  INIT
                   + /REDA   *  YELA   *  YELB   * /Q8    *  INIT
                   +  REDB   *  YELB   *  Q7     *  SENB  *  INIT
                   + /REDA   *  YELA   *  YELB   *  Q7    * /SENA
                             *  INIT

          /Q7     :=  YELA   *  YELB   * /Q8

          /Q8     :=  YELA   *  YELB   *  Q7     * /SENA  * /SENB
                   +  YELA   *  YELB   *  Q7     *  SENA  *  SENB
                   +  YELA   *  YELB   *  Q7     * /Q8
FUNCTION TABLE

Ueberfluessig, da Gleichungen rechnergestuetzt gewonnen werden

++++++++++++  Simulation kann entfallen  ++++++++++++++++++++

DESCRIPTION

Siehe LOGE Datensatz
```

Bild 5. Logische Gleichungen der von LOGE generierten Lösung für die Ampelsteuerung

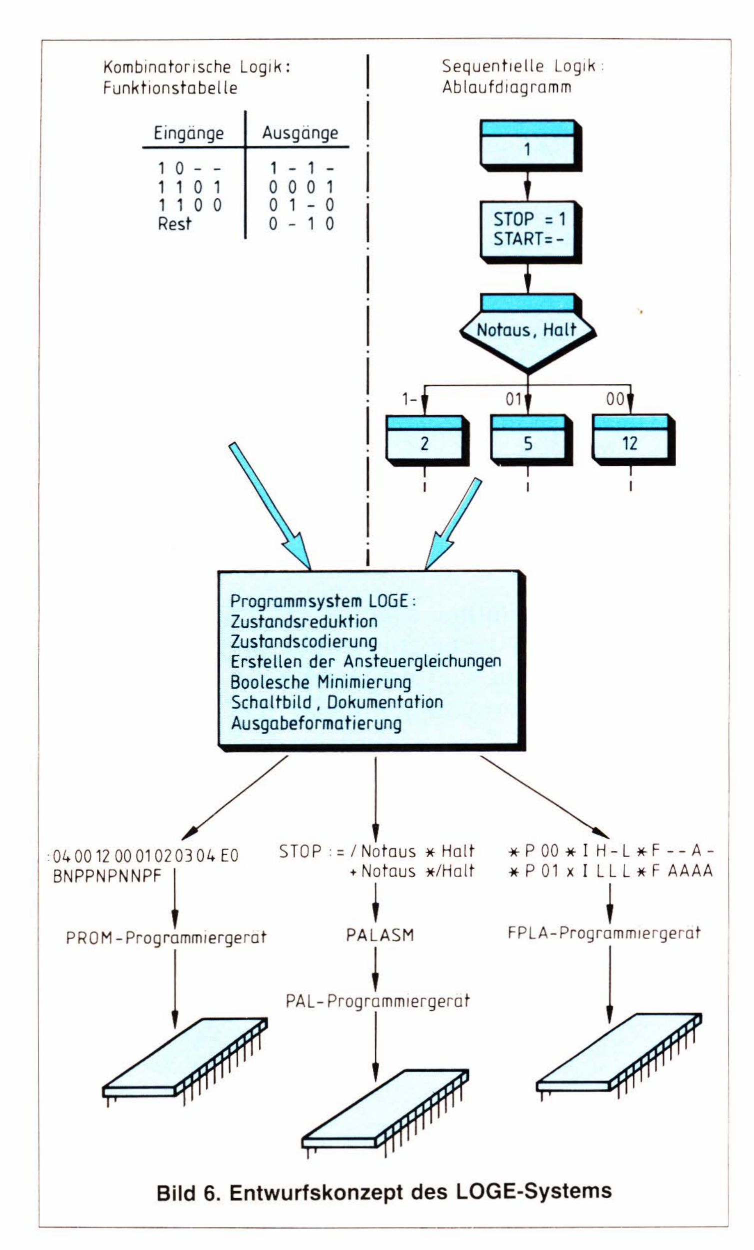


Bild 6. Entwurfskonzept des LOGE-Systems

Gesamtsystem zugrunde liegende Entwurfskonzept anhand von *Bild 6* kurz verdeutlicht werden.

Wesentlichster Bestandteil dieses Konzeptes ist die realisierungsunabhängige Funktionsbeschreibung, die im Falle der kombinatorischen Schaltungen als Funktionstabelle, für die sequentiellen Schaltungen als Ablaufdiagramm ausgeführt wird. Aus dieser Funktionsbeschreibung wird dann die Realisierungsvorschrift für die unterschiedlichen Strukturen vom Programm korrekt erzeugt.

Der Vorteil dieser Vorgehensweise wird am Beispiel der programmierbaren Logikbausteine besonders deutlich: für den Entwerfer stellt sich die Frage: PAL, oder PLA, oder PROM? usw. nicht bereits am Anfang einer Entwicklung, sondern erst an deren Ende. Sollte sich beispielsweise während des Entwurfs zeigen, daß ein Problem mit einem PAL wider Erwarten doch nicht lösbar ist, so ist lediglich ein neuer Rechnerlauf mit *demselben* Eingabedatensatz nötig, um die für ein FPLA erforderlichen Daten zu erzeugen. Mühsames Umformatieren bzw. ein völliger Neuentwurf entfallen also.

Auf diese Weise läßt sich das für eine Problemstellung optimal geeignete Bauelement aus der Vielzahl der möglichen Lösungen, die per Programm in kurzer Zeit erzeugt werden können, auswählen und die technisch und ökonomisch beste Realisierung mit geringem Zeitaufwand finden.

5 Verfügbarkeit

Das Programmsystem LOGE wurde im Rahmen eines vom Bundesministerium für Forschung und Technologie geförderten Projektes an der Universität Karlsruhe unter Leitung von Prof. Dr. H. M. Lipp entwickelt. Wegen der großen Resonanz, die es nach ersten Einsätzen in Industriefirmen fand, wird es seit 1982 durch die Firma Isdata, Karlsruhe, vertrieben und weiterentwikkelt. Das System läuft auf einer breiten Palette von Rechenanlagen, die vom IBM-Personal-Computer bis zum Großrechner reicht.

Literatur

[1] *Schöner, K.*: Programmierbare Logiken in der Praxis. ELEKTRONIK 1982, H. 24, S. 53...58, H. 25, S. 59...65, H. 26, S. 37...43.

[2] *Voldan, W.*: PALASM – eine höhere Assemblersprache für PAL-Bausteine. ELEKTRONIK 1983, H. 1, S. 37...41.

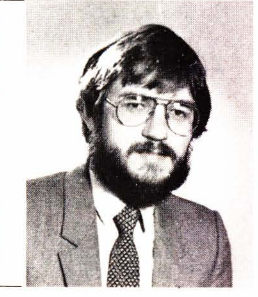
[3] *Lipp, H. M., Grass, W.*: Der rechnergegestützte Entwurf von Steuerwerken. ELEKTRONIK 1977, H. 11, S. 81...85.

[4] Programmsystem LOGE. Benutzerhandbuch, 1982, ISDATA-GmbH, Karlsruhe.

[5] PAL-Handbook 1982, Monolithic Memories, München.

[6] Integrierte programmierbare Logikschaltungen (IFL, Integrated fuse logic) 1982, Valvo, Hamburg.

Dr.-Ing. Günter Biehl ist in Neunkirchen geboren. Er studierte an der Universität Karlsruhe Elektrotechnik mit dem Schwerpunkt Technik der Informationsverarbeitung. Nach seinem Diplom 1976 arbeitete er am Institut für Nachrichtenverarbeitung der Uni Karlsruhe im Rahmen der Projektgruppe „Entwurfsautomatisierung" mit an der Erstellung des Programmsystems LOGE. 1981 promovierte er auf dem Gebiet der mikroprogrammierten Steuerungen. Seit 1982 ist er in der Firma Isdata für CAD-Software zuständig und betreut die Weiterentwicklung von LOGE.





Dr.-Ing. Albrecht Ditzinger, geboren in Stuttgart, studierte in Karlsruhe Elektrotechnik mit dem Schwerpunkt Technik der Informationsverarbeitung. Nach dem Diplom war er für fünf Jahre an der Universität Karlsruhe, Institut für Nachrichtenverarbeitung, im Rahmen einer Projektgruppe für rechnergestützten Schaltungsentwurf tätig, wo er vor allem Pilotanwendungen des Systems LOGE in Zusammenarbeit mit der Industrie durchführte. Seit seiner Promotion Anfang 1982 beschäftigt er sich für die Firma Isdata in Karlsruhe mit rechnergestützten Entwicklungen im Kundenauftrag.

Dipl.-Math. Rüdiger Max

Praktische Erfahrungen mit programmierbaren Logikbausteinen

Programmierbare Logikbausteine haben sich heute in allen Bereichen digitaler Schaltungsentwicklung durchgesetzt. Es gibt eine Vielzahl verschiedener Bausteintypen, die eine Auswahl für die unterschiedlichsten Hardwareanwendungen ermöglichen. Durch die MegaPALs [1] mit über 5000 Gatterfunktionen pro Baustein erreichen programmierbare Logikbausteine Dimensionen, die bisher nur von Gate Arrays beherrscht wurden. Bei der Komplexität dieser Bausteine ist eine Entwicklungssoftware nötig, die eine fehlerfreie und schnelle Entwicklung erst möglich macht. In diesem Aufsatz wird aus der Sicht des Anwenders über die Entwicklung und den Einsatz von PAL-Bausteinen in der Praxis berichtet.

Bei der Entwicklung digitaler Schaltungen hat der Einsatz programmierbarer Logikbausteine (PAL, FPLA, IFL) [1, 4, 5] immer mehr zugenommen. Die große Flexibilität dieser Bausteinstrukturen und die höhere Integrationsdichte gegenüber SSI- oder zum Teil MSI-Chips ermöglichen es dem Entwickler, auch größere Logikkomplexe in nur einem Baustein zu realisieren. Die Entwicklungszeiten vom Logikentwurf bis zum programmierten Baustein sind kurz, das Entwicklungsrisiko ist, etwa im Vergleich zu Gate-Array-Entwürfen, gering. Für die meisten programmierbaren Logikbausteine gibt es mehrere Zweitlieferanten, womit sich derzeit eine gute Liefersituation ergibt.

Neben diesen unbestreitbaren Vorteilen ist der Einsatz von programmierbaren Logikbausteinen mit einem nicht unerheblichen Aufwand verbunden. So benötigt man einen Personal Computer (PC) oder ein Entwicklungssystem mit entsprechender Software sowie ein spezielles Programmiergerät, um die Bausteine entwickeln und programmieren zu können. Bei den Entwicklern muß ein „Know-how" aufgebaut werden, um die Möglichkeiten programmierbarer Logikbausteine voll ausnutzen und die Entwicklungssoftware bedienen zu können.

Hier soll über Erfahrungen berichtet werden, die bei Entwicklung und Einsatz von PAL-Bausteinen in den letzten Jahren gemacht wurden.

Einsatz

Bei der Neuentwicklung einer Baugruppe wird die gesamte Logik im „Top-Down"-Entwurf in Funktionsblöcke aufgeteilt, die in mehreren Strukturebenen verfeinert werden, bis sich überschaubare Logikkomplexe ergeben haben. Läßt sich ein Teil dieser Logikkomplexe durch einzelne PAL-Bausteine realisieren, so wird die im Entwurf erzeugte übersichtliche Struktur auch in der Hardware beibehalten.

Um die PAL-Bausteine zu entwickeln, sollte man sie als „Black Box" betrachten und ganz abstrakt Schaltnetze (kombinatorische Logik) und Schaltwerke (sequentielle Logik) entwerfen, die je nach verwendetem PAL-Bausteintyp nur von den Ein-/Ausgängen, den Produkttermen und der Anzahl der Zustände abhängig sind. Schon in einem Medium-PAL (z. B. 16L8, 16R6, 20L8, 20R6) [1] lassen sich auf diese Weise beachtliche Logikkomplexe integrieren und je nach Anwendung mehrere SSI- oder MSI-Chips ersetzen.

Es ist zu empfehlen, sich zumindest in einem Teil der auf einer Baugruppe eingesetzten PALs eine Hardwarereserve zu lassen. Auf diese Weise hat man die Möglichkeit, bei Logikänderungen die PAL-Schaltnetze oder -Schaltwerke auszuwechseln, ohne die Hardwarestruktur verändern oder Verbindungen auf Baugruppe trennen bzw. hinzufügen zu müssen.

Es wurden auch gute Erfahrungen bezüglich des elektrischen Verhaltens der PALs bei der Kombination mit Bausteinen der Familien „Low Power Schottky" und „FAST" gemacht.

Ein anderer Anwendungsfall für PALs ist gegeben, wenn eine vorhandene und voll belegte Baugruppe in ihrem Funktionsumfang erweitert werden muß. Um die nötigen Einbauplätze für die Funktionserweiterung zu bekommen, versucht man zunächst, Gatterlogik einschließlich Flipflops in einem Logikkomplex zusammenzufassen, der bezüglich der Ein- und Ausgänge in

einem PAL zu realisieren wäre und möglichst noch in dem Bereich liegt, der sowieso verändert werden soll. Die Logikverdichtung ist besonders effektiv, wenn es auch noch gelingt, die alte Gatterlogik zu optimieren. Erfahrungsgemäß macht dies jedoch die größten Schwierigkeiten, da unter Umständen die Funktionserweiterung nicht vom Erstentwickler der Baugruppe durchgeführt wird. In diesem Fall kann man nur sehr schwer feststellen, ob die Logik Redundanzen enthält, die sich für die Optimierung anbieten.

PALs sind teurer als SSI- oder MSI-Bausteine, selbst wenn man einen höheren Integrationsfaktor zugrunde legt. Erhebliche Einsparungen ergeben sich durch den Einsatz von PAL-Bausteinen jedoch immer dann, wenn auf einer Baugruppe nicht mehr genügend freie Einbauplätze zur Realisierung einer gewünschten Funktion zur Verfügung stehen und man sonst gezwungen wäre, eine größere oder sogar eine zusätzliche Baugruppe zu entwickeln.

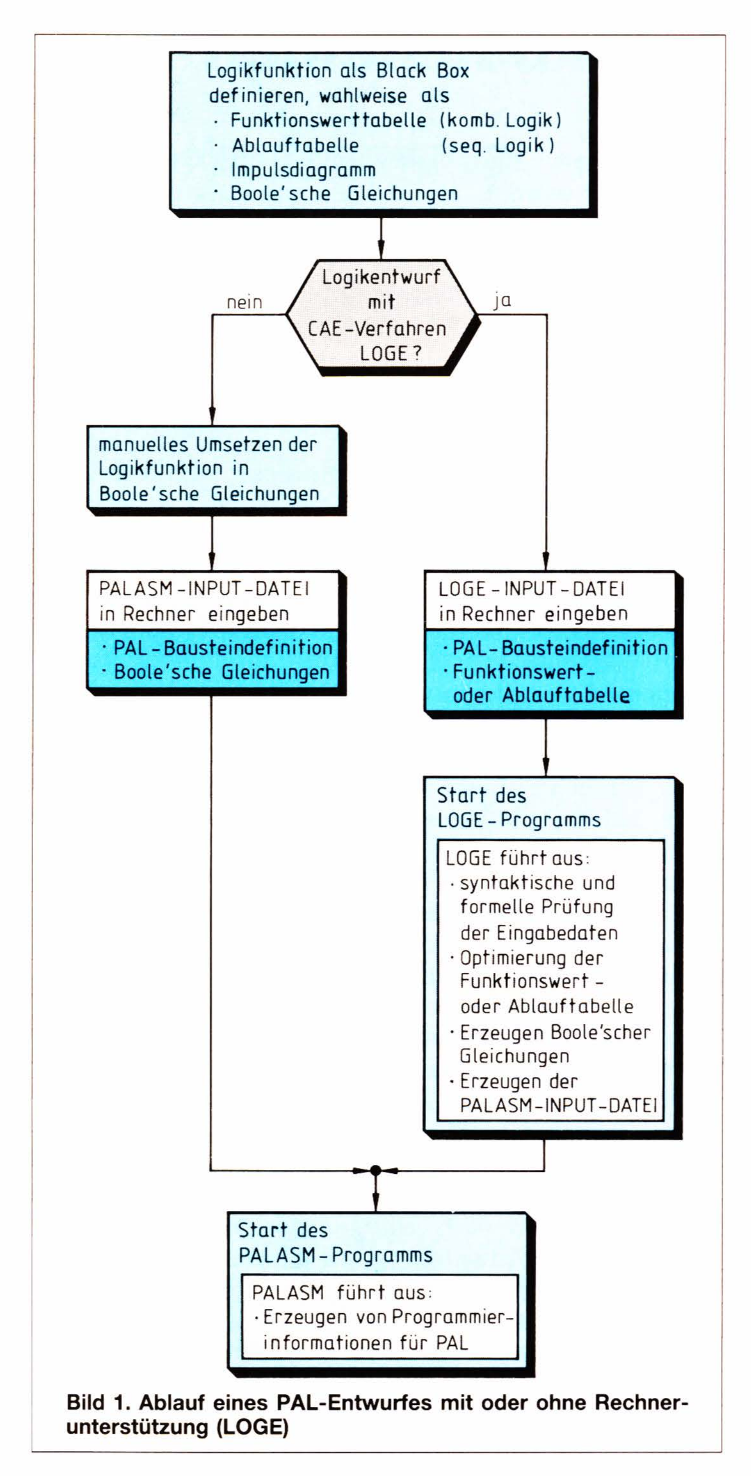


Bild 1. Ablauf eines PAL-Entwurfes mit oder ohne Rechnerunterstützung (LOGE)

Entwicklungsverfahren

Mit den derzeitig erhältlichen Medium-PALs lassen sich Schaltnetze mit z. B. 16 Eingängen und 6 Ausgängen (20L8) [1] oder Schaltwerke mit z. B. 10 Eingängen, 2 Ausgängen und 8 Zustandsvariablen (20RS8) [1] entwickeln. Je nach PAL-Typ können beim 20L8 bis zu 7 und beim 20RS8 bis zu 16 Produktterme pro Ausgang benutzt werden, um die gewünschte Logikfunktion zu definieren. Mit der zur Verfügung stehenden Zahl an Ein- und Ausgängen sind daher schon mit Medium-PALs sehr komplexe Schaltnetze und Schaltwerke realisierbar. Erfahrungsgemäß muß der Entwickler jedoch in vielen Anwendungsfällen seine Logikfunktion optimieren, um mit der begrenzten Produkttermzahl pro Ausgang zurechtzukommen. Die fehlerfreie Optimierung der Produktterme ist ab einer bestimmten Größenordnung und in einem sinnvollen Zeitaufwand nur noch mit einem CAE-Verfahren zu bewältigen.

Mit den MegaPALs 32R16 und 64R32 [1] lassen sich noch erheblich größere Logikkomplexe in einem Baustein integrieren. Sie sind jedoch, mit Ausnahme von sehr symmetrischen Strukturen (z. B. Zähler, Multiplexer), ohne ein rechnergestütztes Entwicklungsverfahren nicht zu entwickeln.

Es gibt derzeit mehrere Softwareprogramme (z. B. ABEL, CUPL, LOGASM, LOGE, PALASM), welche die Entwicklung von PAL-Bausteinen und insbesondere die Optimierung von Logikfunktionen unterstützen. Der Optimierung von PAL-Logik ist aus mehreren Gründen eine große Bedeutung beizumessen. Im Vordergrund steht der Kostenaspekt; denn eine gewünschte Logikfunktion ist in dem jeweils kleinsten und damit am besten ausgenutzten PAL am billigsten zu realisieren. Eine Optimierung kann außerdem die nötige Hardwarereserve liefern, die für eine spätere Logikänderung eventuell nötig ist. Je weniger Produktterme pro PAL-Ausgang benutzt werden, um so geringer ist die Zahl der vom Entwickler zu definierenden Testvektoren, und um so geringer ist auch der Stromverbrauch.

Im folgenden wird die PAL-Entwicklung ohne Optimierungsprogramm der PAL-Entwicklung mit einem CAE-Verfahren, in diesem Fall dem Programm LOGE [3], gegenübergestellt.

PAL-Entwicklung ohne rechnerunterstützte Optimierung

In vielen Fällen kann man die PAL-Logik unmittelbar als Boole'sche Gleichungen angeben, die dann vom PAL-Assembler (PALASM) [2] zu Programmierinformationen weiterverarbeitet werden können (*Bild 1*). Aller-

dings ist dies nur für relativ kleine, überschaubare Logikfunktionen (z. B. Decodierlogik) oder für sehr symmetrische Logikstrukturen (z. B. Multiplexer, Zähler) möglich.

Es gibt zwar für den Handentwurf geeignete Optimierungsmethoden, wie z. B. das Verfahren von Karnaugh und Veitch; sie sind allerdings bei mehr als sechs Variablen sehr unübersichtlich und fehleranfällig. Daher lassen sich komplexere Schaltnetze oder Schaltwerke mit mehr als sechs Eingangs- oder Zustandsvariablen nur mit erheblichem Zeitaufwand oder gar nicht ohne ein Optimierungsprogramm entwickeln.

PAL-Entwicklung mit LOGE

Dem Programm LOGE wird die gewünschte PAL-Logik als Funktionswerttabelle für Schaltnetze oder als Ablauftabelle für Schaltwerke angegeben. LOGE überprüft dann die syntaktische und formelle Richtigkeit der Eingabedaten und erzeugt in kurzer Zeit ein optimiertes Schaltnetz oder Schaltwerk in Form von Boole'schen Gleichungen. Sollte die von LOGE optimierte Funktion nicht in dem ausgewählten PAL-Typ zu realisieren sein, so muß der Entwickler die Funktionswert- oder Ablauftabelle verändern (z. B. durch Auswahl einer geeigneten Zustandscodierung oder durch Einschränkung von Redundanzen in der Funktionswerttabelle), um nach einem erneuten Aufruf des Programms zu einem eventuell besseren Optimierungsergebnis zu kommen. Auf diese Weise wird der PAL-Baustein am Terminal unter Umständen erst nach mehreren Programmdurchläufen entwickelt, was aber wegen der geringen Programmlaufzeit immer noch sehr schnell geht.

Nachdem man die endgültige PAL-Logik entwickelt hat, kann man sich automatisch von LOGE den Eingabedatensatz des PAL-Assemblers erzeugen lassen und hat damit einen geschlossenen Weg vom LOGE-Eingabedatensatz zu den PAL-Programmierinformationen.

Der Editieraufwand für das Erstellen des Eingabedatensatzes für PALASM an einem PC oder Entwicklungssystem ist, verglichen mit dem Erstellen des Eingabedatensatzes für LOGE, nur unwesentlich geringer.

Neben der Zeitersparnis für die Entwicklung und fehlerfreien Realisierung einer PAL-Logik gibt es noch weitere Gründe, die für den Einsatz eines CAE-Verfahrens sprechen. Da die Funktionswert- oder Ablauftabelle immer die Basis einer PAL-Logik bilden, ist automatisch eine standardisierte Dokumentation des jeweiligen Logikkomplexes gegeben. Zusätzlich kann jede Zeile der Funktionswert- oder Ablauftabelle im LOGE-Eingabedatensatz mit Kommentaren versehen werden, die auf

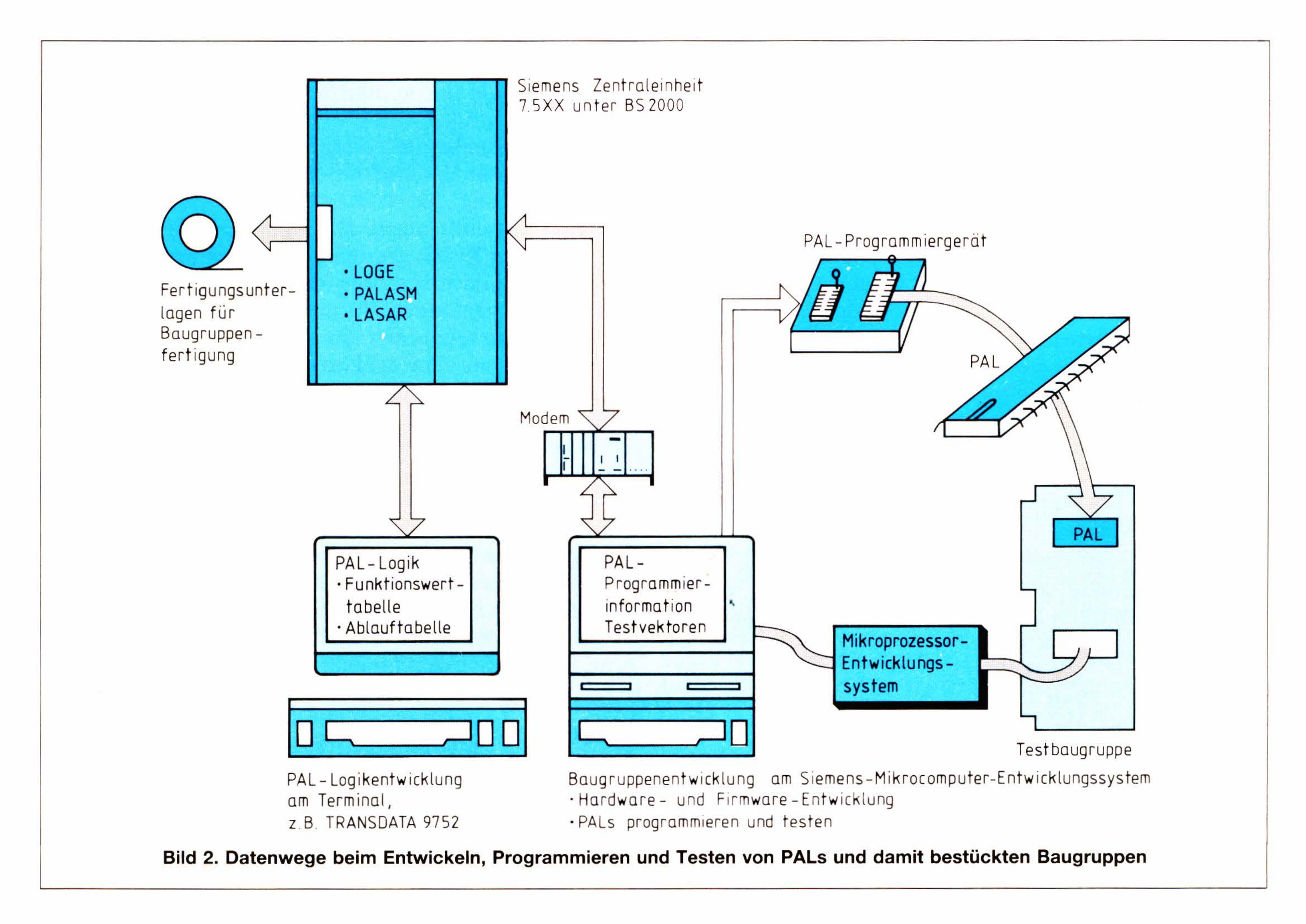
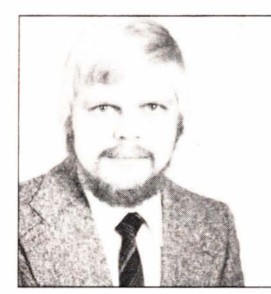


Bild 2. Datenwege beim Entwickeln, Programmieren und Testen von PALs und damit bestückten Baugruppen



Dipl.-Math. Rüdiger Max, geboren in Oker/Harz, studierte angewandte Mathematik und Informatik an der Universität Hamburg mit Schwerpunkt Minimierungsverfahren für nichtlineare Gleichungssysteme. Seit 1980 ist er Entwickler bei der Siemens AG, München, im Bereich Datentechnik. Er besitzt mehrjährige Erfahrung mit Entwicklung und Einsatz programmierbarer Logikbausteine.

diese Weise mit in die Dokumentation eingehen. Daher können sich auch andere Entwickler schnell in die Logik einarbeiten und eventuelle Wartungsarbeiten oder Funktionsänderungen vornehmen.

Programmierter Baustein

In *Bild 2* sind in einem konkreten Anwendungsbeispiel die Datenwege dargestellt, die für den Einsatz von PAL-Bausteinen geschaffen werden müssen.

Die Programme LOGE und PALASM laufen hier unter dem Betriebssystem 2000 auf einer Siemens-Zentraleinheit. Die Entwicklung der PAL-Logik wird an einem Terminal durchgeführt. Die Programmierinformationen der entwickelten PAL-Bausteine gelangen über ein Modem an ein Entwicklungssystem und von dort über eine RS-232C-Schnittstelle an ein PAL-Programmiergerät. Wie man in Bild 2 sieht, wird hier dieselbe Anordnung für die Programmierung der PALs benutzt, mit der auch die übrige Hardware und Firmware einer Baugruppe entwickelt und getestet wird. Die Datensätze der auf der Zentraleinheit entwickelten PAL-Bausteine lassen sich dort direkt für die Fertigungsunterlagen sowie für die Baugruppensimulation weiterverarbeiten.

Baugruppenfertigung

Um Baugruppen von höchster Qualität fertigen zu können, sind u. a. umfangreiche Prüfungen aller Bausteine einer Baugruppe erforderlich. PALs sind hier aufwendiger zu handhaben als SSI- oder MSI-Bausteine.

Nach der Programmierung eines PALs verifiziert das Programmiergerät die Verbindungsmatrix des Bausteins (AND-Array) [2]. Damit ist jedoch noch nicht sichergestellt, daß der Baustein auch fehlerfrei funktioniert, weil hierbei keine vollständige Prüfung aller Leitungswege des Bausteins stattfindet. Es muß vielmehr mit Hilfe von Testvektoren eine Funktionsprüfung des PALs an einem Bausteintestgerät durchgeführt werden. In beschränktem Maße ist diese Prüfung auch an einigen PAL-Programmiergeräten möglich.

Das manuelle Aufstellen der Testvektoren ist insbesondere für Schaltwerke, die mit PAL-Bausteinen ohne Preload-Funktion [1] realisiert wurden (z. B. 16R6, 20R8), kompliziert und zeitaufwendig. Zum Prüfen dieser PALs sind lange Testvektorsequenzen erforderlich, die jeweils vom Grundzustand ausgehend alle Statusübergänge des Schaltwerkes austesten. Auf der Siemens-Zentraleinheit (Bild 2) sind die zur Funktionsprüfung nötigen Testvektoren mit dem Programm LASAR [6] automatisch generierbar.

Neben der funktionellen Fehlerfreiheit sind, wie bei anderen Bausteinen auch, die elektrischen Parameter eines PALs zu überprüfen. Die programmierten PAL-Bausteine werden dann auf der bestückten Baugruppe im Zusammenhang mit der übrigen Bausteinumgebung von Baugruppentestgeräten und mikroprogrammierten Selbsttests geprüft.

Zukünftige CAE-Tools

Die Softwarepakete zur Entwicklung von PAL-Bausteinen sind in den letzten Jahren immer weiter verbessert worden. Nachfolgeprogramme des PALASM werden eine noch komfortablere Logikbeschreibung von PAL-Bausteinen ermöglichen. Bei dieser durchaus wünschenswerten Entwicklung sollte man aber bedenken, daß sich die meisten PAL-Logikkomplexe (insbesondere Schaltwerke oder unsymmetrische Logikstrukturen) nicht so einfach als Boole'sche Gleichungen oder n-dimensionale Gleichungsvektoren hinschreiben lassen. Es gibt zudem noch Schwachstellen in allen heute zur PAL-Entwicklung einsetzbaren CAE-Programmen, deren Verbesserung den Anwendern sicherlich mehr zugute käme als eine komfortablere Logikbeschreibungssprache.

So gibt es leider noch kein Programm, das automatisch eine optimale Zustandscodierung für PAL-Schaltwerke berechnet. Die Wahl der Zustandscodierung hat aber einen erheblichen Einfluß auf die Zahl der Produktterme, die zur Realisierung eines Schaltwerkes benötigt werden. Eine Lösung dieses Problems würde daher eine bessere Ausnutzung der Register-PALs ermöglichen.

Des weiteren könnten die Optimierungsprogramme automatisch aus den optimierten Funktionswert- bzw. Ablauftabellen die Testvektoren erzeugen, die zur Funktionsprüfung der PAL-Bausteine unbedingt erforderlich sind. Man kann sicherlich noch weitere Verbesserungsvorschläge für CAE-Verfahren anbringen, die eine Entwicklung der PAL-Bausteine erleichtern würden.

Für den Entwickler ist es jedenfalls wünschenswert, ein CAE-Verfahren zu haben, das ihm, ausgehend von der Funktionswert- oder Ablauftabelle, die PAL-Programmierinformation erzeugt, ohne daß er sich um Detailprobleme, wie z. B. Optimierung, Zustandscodierung oder Testvektorerzeugung, kümmern muß.

Literatur

[1] LSI Databook Sixth Edition. Monolithic Memories.
[2] PAL Handbook Third Edition. Monolithic Memories.
[3] Benutzerhandbuch Programmsystem LOGE, Version 3.1, ISDATA.
[4] Integrierte programmierbare Logikschaltungen. Datenbuch 1984. Valvo.
[5] Programmable Array Logic, Handbuch, Advanced Micro Devices.
[6] LASAR Users Guide 4.0. Electronic Test Services.

Willibald Voldan

Systementwicklung mit „MegaPALs“

PALs bisheriger Komplexität mit 50 bis 300 Gatterfunktionen im 20- und 24-Pin-Gehäuse wurden bisher fast ausschließlich als Ersatz von herkömmlicher SSI/MSI-Logik verwendet; die relativ geringe Komplexität und limitierte Anzahl von Ein- und Ausgängen ermöglichten nur den bedingten Einsatz für „intelligente“ logische Funktionen. Als vor einiger Zeit der Erfinder der PAL-Familie, John Birkner, und MMI's Design Manager Sing Wong zu einer Besprechung zusammentrafen, wurde dabei die nächste Generation programmierbarer Logik definiert, die in Größe und Komplexität mit den bisherigen Gate Arrays gleichziehen sollte. Das Ergebnis dieser Besprechung war eine Familie von PAL-Bauelementen mit mehreren 1000 Gatterfunktionen und zusätzlichen programmierbaren Eigenschaften. Diese „MegaPALs“ im 40- und 84-Pin-Gehäuse ermöglichen es, auch mit den erweiterten logischen Möglichkeiten, wesentlich komplexere Funktionen (Controller, I/O-Funktionen, simple µP-Strukturen usw.) zu übernehmen. Über die Programmierbarkeit auch dieser großen PAL-Bausteine sind jetzt beliebige, bisher nicht realisierbare logische Funktionen in unbegrenzter Vielfalt möglich geworden.

Die Familie der MegaPALs besteht zur Zeit der Drucklegung (Mitte 1985) aus zwei Typen.

Das PAL32R16 im 40-Pin-DIP- und im 44-Pin-Chip-Carrier-Gehäuse (Leadless Ceramic und Molded Plastic) besitzt eine verfügbare Gatterkomplexität von über 1500 Gatterfunktionen (siehe die Berechnung der Komplexität in dem Artikel: Berechnung der Gatterkomplexität programmierbarer Logik) und über 8000 Sicherungselemente.

Das PAL64R32 wird im 84-Pin-Leadless-Chip-Carrier (Keramik) und im 88-Pin-Grid-Array gefertigt (*Bild 1*). Dieser Baustein weist eine Komplexität von 5696 frei programmierbaren Gatterfunktionen auf und beinhaltet über 32 000 Sicherungen, um die logischen Verbindungen zwischen den 64 Eingängen und 32 Ausgängen mit Rückkopplung herzustellen.

Wie bei allen PALs finden wir auch bei diesen Elementen eine globale logische Verknüpfbarkeit, d. h. jeder Eingang ist beliebig oft zu jedem Ausgang zu schalten.

Auch diese großen Bauelemente realisieren die bereits bekannte Struktur der herkömmlichen PALs, die UND-ODER-Konfiguration mit D-Flipflop und interner Rückkopplung. Darüber hinaus konnten in diese beiden Bausteine zusätzliche programmierbare logische Eigenschaften mit integriert werden, die bisherige Einschränkungen in der PAL-Konfiguration weitgehend ausmerzen. Die „logische Tiefe“ der Bauelemente ist über das besondere Verfahren des „Product Term Sharing“ auf 16 mögliche Produkt-Terme pro Ausgang erhöht worden. Jeder Ausgang ist individuell, also unabhängig von den anderen, auf „Low- bzw. High-Aktiv“ programmierbar, und die Register besitzen eine „Bypass“-Schaltung, die es erlaubt, an Stelle jedes der 32 Ausgangsregister auch einen kombinatorischen Ausgang zu erzeugen.

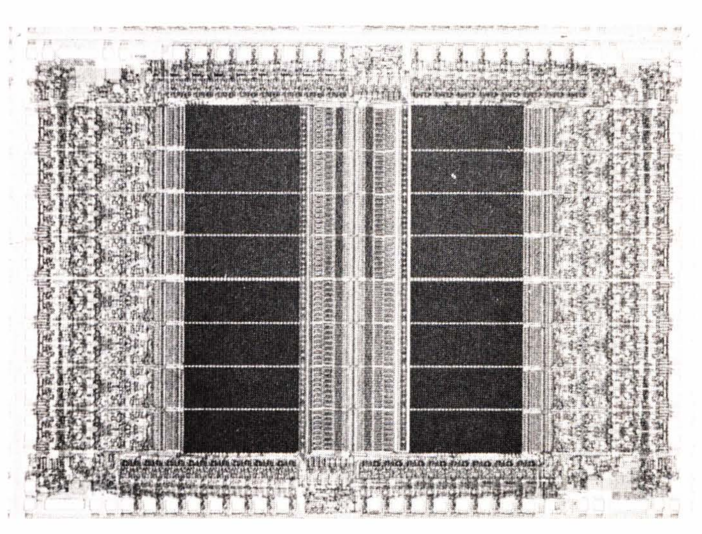


Bild 1. Die Chipfotografie des PAL64R32 offenbart die Ähnlichkeit mit der Struktur von Speichern (RAM, PROM). Dieses MegaPAL nimmt eine Chipfläche von fast 80 000 mil² in Anspruch (1 mil = 1/1000 Zoll)

Das PAL32R16 ist in zwei Bänke von je acht Registern unterteilt. Die „Bypass-Schaltung“ ist für jede Bank individuell durchführbar. Das PAL64R32 besitzt 4 × 8 dieser Registerbänke mit D-Flipflops (*Bild 2* und *3*).

Jede dieser Register-Bänke hat ihren eigenen synchronen Takteingang, einen für alle Ausgänge dieser Bank zugewiesenen Freigabeeingang (ENABLE) und einen weiteren Steuereingang für das asynchrone Vorladen der Register (PRELOAD).

Das PAL64R32 mit 84 Anschlüssen besitzt noch zusätzlich für jede seiner vier Registerbänke je einen asynchronen Rücksetz-Eingang (ASYNCHRONOUS RESET).

Im *Bild 4* ist eine komplette Logikzelle eines PAL32R16 bzw. PAL64R32 dargestellt. Damit werden nachfolgend die besonderen Möglichkeiten dieser Anordnung beschrieben.

Das „Product Term Sharing“

Die Architektur programmierbarer Logikbausteine besteht im wesentlichen aus zwei Arrays, nämlich den UND-ODER-Matrizen. Die Eingänge zum UND-Array können zu jedem der vorhandenen UND-Gatter beliebig oft ein Produkt bilden. Daher nennt man diese Anordnung der UND-Gatter auch Produkt-Terme. Das logische Produkt wird dann über ein festgelegtes bzw. programmierbares ODER-Array an den Ausgang geführt. Die Anzahl der Produkt-Terme pro Ausgang bestimmt daher maßgeblich die logische Tiefe (ODER-Verknüpfungen) eines PAL-Bauelementes.

Die Summe der Produkt-Terme (PT) in einem Bauelement und damit auch pro Ausgang kann nicht beliebig erhöht werden, da sie maßgeblich am Leistungsverbrauch des Bauelementes beteiligt ist (die Anzahl der Produkt-Terme geht fast linear in den Stromverbrauch ein). Um aber trotzdem die Zahl der PT pro Ausgang erhöhen zu können, verwirklichte Monolithic Memories in seinen neuesten PALs die Eigenschaft des „Product Term Sharing“. Dabei werden über ein weiteres programmierbares Feld jetzt immer 16 PT (bisherige PALs nur 8) zwei nebeneinanderliegenden Ausgängen zugeordnet. Über Programmierung der Sicherungen in diesem Array kann nun jeder der beiden Ausgänge eine

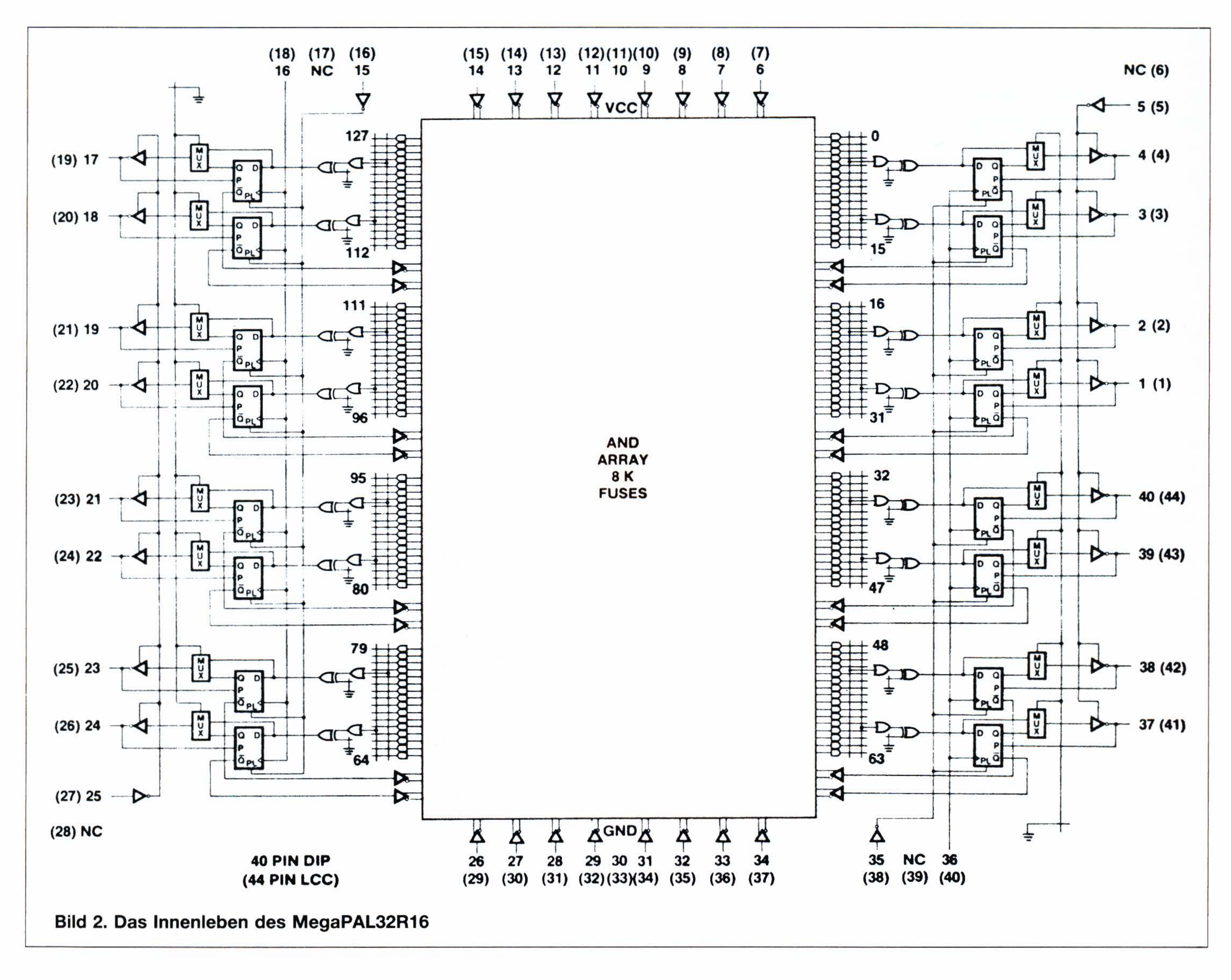


Bild 2. Das Innenleben des MegaPAL32R16

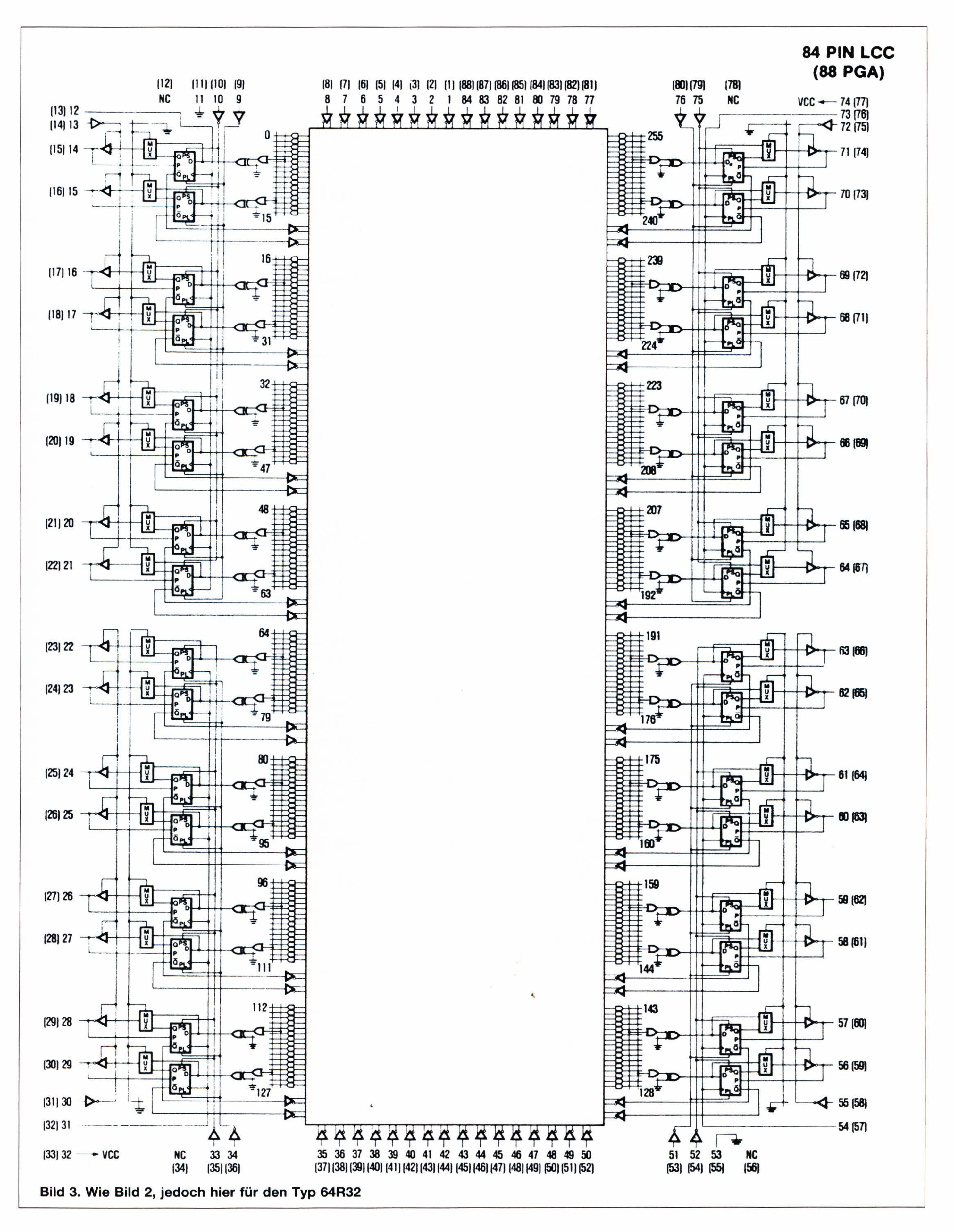


Bild 3. Wie Bild 2, jedoch hier für den Typ 64R32

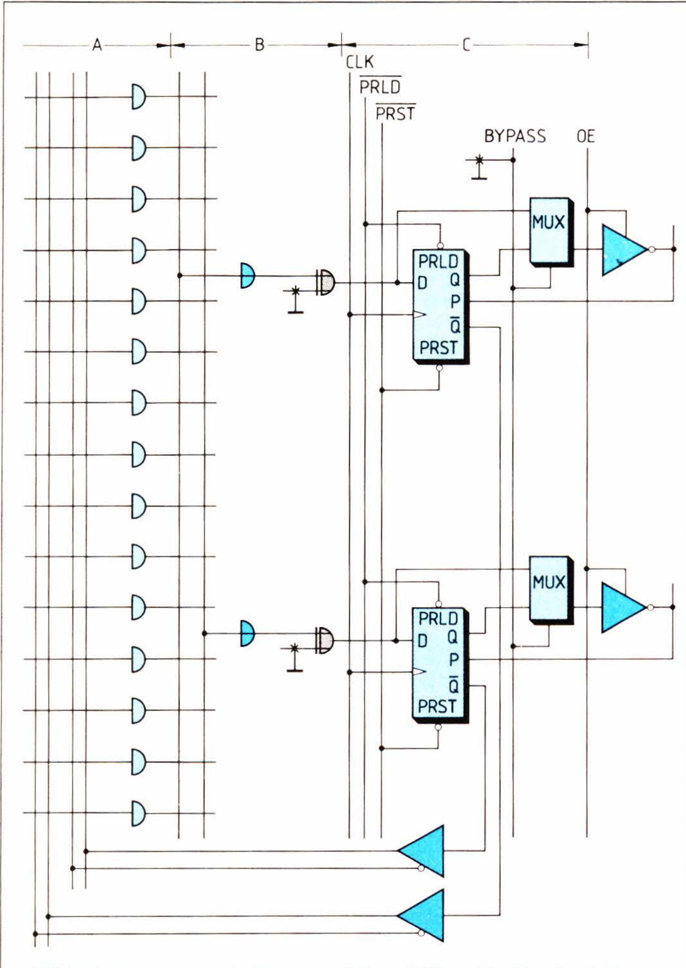


Bild 4. Ausgangsstufe eines MegaPAL mit „Product Term Sharing", „Bypass Register", „Preload" und programmierbarer Polarität des Ausgangssignals

Anzahl zwischen 0 und 16 PT erhalten, also z. B. 1:15, 2:14, 8:8, 16:0 usw. Das bedeutet, daß damit ein Produkt-Term entweder dem einen Ausgang oder dem anderen, oder beiden, oder auch keinem zugeordnet werden kann. Mit dieser Variation von PT lassen sich wesentlich komplexere logische Gleichungen in diesen PALs unterbringen. Insgesamt besitzt das PAL32R16 128 und das PAL64R32 256 Produkt-Terme. Die Eigenschaft des „Product Term Sharing" kann darüber hinaus auch als „Product Term Editing" bezeichnet werden.

Wie oben beschrieben, kann man mit dieser Architektur einen oder mehrere PT auch keinem der beiden Ausgänge zuordnen. Dabei werden eben beide Sicherungen im programmierbaren ODER-Array des jeweiligen Ausgangspaares programmiert und z. B. während der Entwicklung ein bereits programmierter oder nicht mehr benötigter Produkt-Term vom Gesamt-Array abgetrennt und an dessen Stelle ein anderer verwendet.

Dieses Editieren eines PAL-Bauelementes kann fortgeführt werden, solange weitere unbenutzte Produkt-Terme für diesen Ausgang vorhanden sind. Diese logische Eigenschaft werden sich auch zukünftige Programmiergeräte zunutze machen. Läßt sich eine bestimmte Sicherung in einem Array nicht programmieren und sind noch weitere PT für diesen Ausgang vorhanden, so trennt eben der Programmer die beiden „Sharing Fuses" dieses PT und verwendet den nächsten.

Ein weiterer Vorteil dieser Eigenschaft liegt darin, daß mit dem Abtrennen von Produkt-Termen auch der Stromverbrauch des gesamten Bauelementes sinkt. Nachdem die Logik einmal fertig erstellt ist, können sämtliche Produkt-Terme, die im Logikpattern nicht verwendet sind, vom Stromverbrauch abgekoppelt werden.

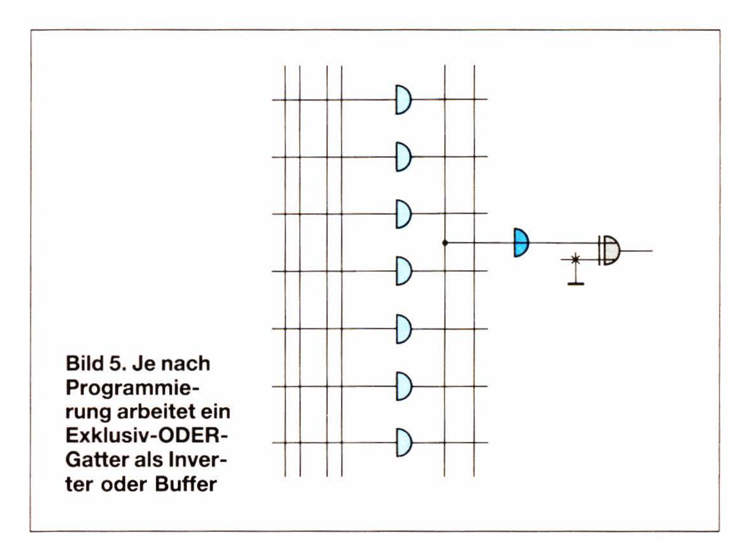
Programmierbare Ausgangspolarität

Nach der UND-ODER-Anordnung wird das Signal über ein Exklusiv-ODER-Gatter geführt, dessen zweiter Eingang über eine Sicherung an Masse angeschlossen ist.

Bleibt diese Sicherung bestehen, übernimmt das EX-ODER-Gatter lediglich Buffer-Funktion. Nach der Programmierung dieser Sicherung wird das EX-ODER zum Inverter; das Signal wird als logisches Komplement an den Ausgang geführt (*Bild 5*).

Das „Bypass Register"

Jede Ausgangszelle der MegaPALs kann das Signal über das D-Register oder kombinatorisch an den Ausgang führen. Die Entscheidung trifft ein 2:1-Multiplexer, dessen Steuerung über eine weitere Sicherung erfolgt. Eine Sicherung ist verantwortlich, jeweils eine Bank von acht Ausgängen kombinatorisch oder über die zugehörigen Register zu schalten. Ist diese Sicherung intakt, laufen die Signale vom kombinatorischen Array in den D-Eingang des Registers. Ist diese Sicherung programmiert, und wird das Signal über den Multiplexer am Register vorbeigeführt, kann das Flipflop trotzdem als „Buried Register" weiterverwendet werden. Diese Funktion ist besonders für Steuerungsaufgaben außerordentlich wichtig (*Bild 6*).


Bild 5. Je nach Programmierung arbeitet ein Exklusiv-ODER-Gatter als Inverter oder Buffer

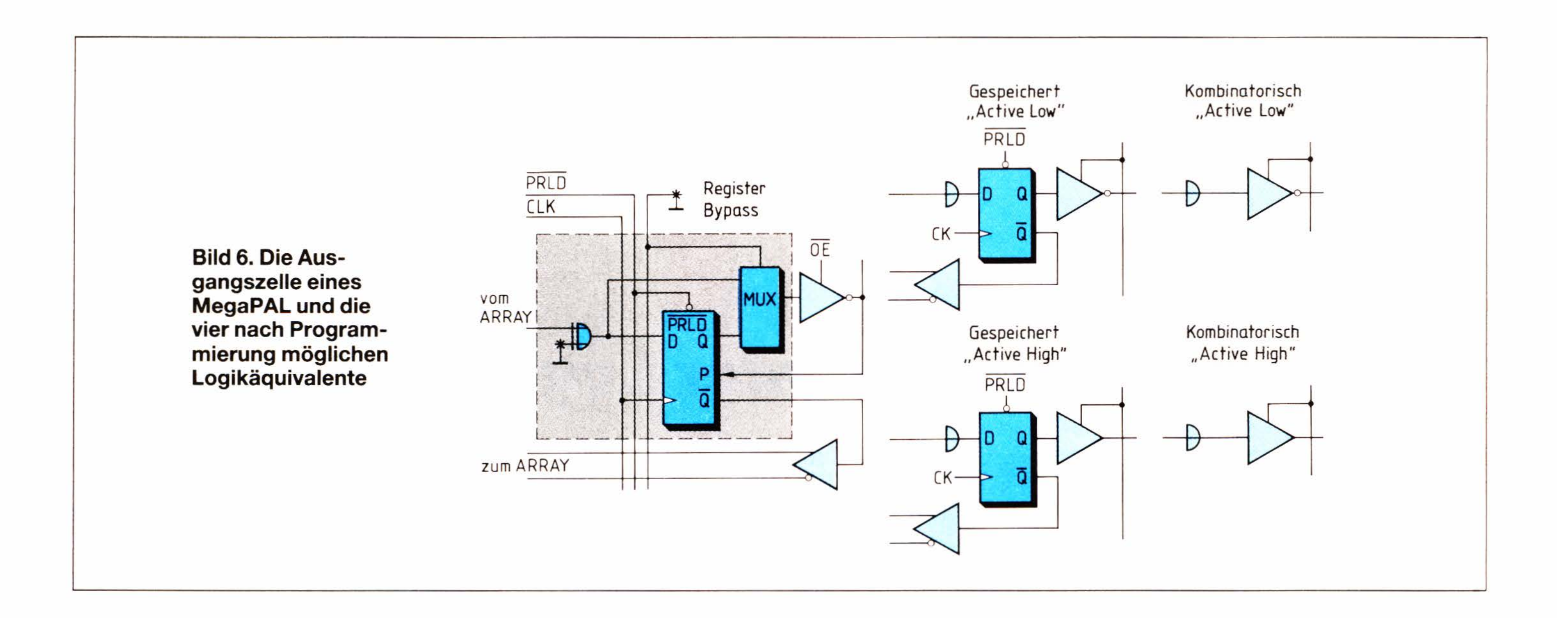


Bild 6. Die Ausgangszelle eines MegaPAL und die vier nach Programmierung möglichen Logikäquivalente

„Register Preload"

Mit PALs und insbesondere MegaPALs können sehr komplexe kombinatorische und sequentielle Schaltungen (Statusmaschinen) realisiert werden. Programmierbare Logik ermöglicht noch dazu eine beliebige Vielfalt von verschiedenen Funktionsmustern, die auch testfähig sein müssen.

Um eine sequentielle Schaltung mit der Komplexität eines MegaPALs nach der Programmierung mit einem individuellen Logikpattern funktionell testen zu können, ist es möglich, über eine spezielle „Preload"-Einrichtung jedes Register mit einem beliebigen Logikpegel vorzuladen. Damit kann z. B. eine Schaltung auf wichtige Übergangsfunktionen geprüft werden, um sicherzustellen, daß eine sequentielle Schaltung nach einer unerlaubten Sprungfunktion auch wieder in die richtige Sequenz zurückkehrt, bevor das Bauelement in die Gesamtschaltung eingesetzt wird.

Obwohl PAL-Bausteine über eine „Power-up-Reset"-Einrichtung verfügen (nach dem Anlegen der Versorgungsspannung nehmen die Register automatisch den inaktiven Zustand ein), wäre es schwierig, das Bauelement in einen bekannten logischen Zustand zu bringen. Mit 32 Registern kann es im schlimmsten Fall 2^{32} (ca. $4 \cdot 10^9$) Zustände benötigen, bevor ein gewünschter Status erreicht wird.

Mit der „Preload"-Einrichtung können auch Testzeiten wesentlich verringert werden. Sie funktioniert über einen besonderen Steuereingang, der für jede Registerbank individuell eingerichtet ist. Nachdem die Ausgänge mit dem „Enable"-Eingang in den Three-State-Zustand gebracht sind, werden mit einem „Low-Pegel" am „Preload"-Eingang die Daten, die am Ausgang der jeweiligen Registerbank anliegen, in die Flipflops übernommen.

Außer für Prüfzwecke ist diese Eigenschaft natürlich auch noch für logische Schaltungen verwendbar, um z. B. Zustände der Außenwelt (Bus-System) über ein Vorladen in den Registern zwischenzuspeichern bzw. zu beurteilen. Das ist aber sicher nur ein Beispiel, wozu die Preload-Schaltung neben dem Testen noch verwendet werden kann.

Wie bei herkömmlichen PAL-Bausteinen lassen sich auch die MegaPALs über die Programmierung einer letzten Sicherung vor dem Auslesen schützen, um damit die Schaltung vor dem Duplizieren zu sichern.

Abschließend sei noch darauf hingewiesen, daß es die MegaPALs auch in einer Maskenversion (HAL) gibt. Diese bewährt sich dort, wo eine vorgegebene Logik in großen Stückzahlen und ohne Änderungen benötigt wird.

Willibald Voldan

Testen programmierbarer Logik

Die Problematik der Funktionsprüfung digitaler Schaltungen entfernt sich generell von der Leiterplatte hin zum LSI-Baustein. Mit steigender Integration logischer Bauelemente werden die Leiterbahnen, die auf einer Printplatte SSI/MSI-Gatter und -Flipflops verbinden, auf Silizium übertragen. Die Entflechtung der Platine erfolgt bereits auf dem Chip. Das bedeutet, daß höher entwickelte Testprogramme vorhanden sein müssen, um ohne Zuhilfenahme interner Testpunkte, wie es noch auf der Leiterplatte möglich war, zu testen. Die Funktionsprüfung hochintegrierter Bauteile ist kein neues Problem. Standard-LSI und -VLSI-Bauelemente wie Mikroprozessoren und deren Peripheriebausteine werden seit einiger Zeit mehr oder weniger erfolgreich getestet. Der Unterschied liegt darin, daß mit programmierbarer Logik in sehr kurzer Zeit viele verschiedene Logik-Pattern generiert werden können.

Vorgehensweise

Testprogramme für LSI-Bausteine bestehen aus einem Parameter- und einem Funktionstest. Die Messung der Parameter wird vom Hersteller vorgenommen; die Funktionsprüfung bleibt dagegen meist dem Anwender überlassen. Dieser logische Funktionstest wird über Testvektoren durchgeführt, die den Pin-Namen und die logische Bedingung (LOW, HIGH, CLOCK, THREE-STATE usw.) definieren.

Die Generierung von Testvektoren für einen LSI-Baustein ist aufwendig und zeitraubend, weshalb die Erstellung einem Rechner übertragen wird. Aber selbst diese Testprogramme benötigen Grundvektoren, die der Entwickler der jeweiligen Schaltung selbst zur Verfügung stellen muß.

Wird eine Neuentwicklung der Serienproduktion überstellt, so hat die Testabteilung zu bestimmen, wie und was zu testen ist, um die einwandfreie Funktion des Subsystems sicherzustellen. Meist geht gerade an diesem Punkt viel Information verloren. Der Entwickler wendet sich einem neuen Design zu, und der Testingenieur wird mit der Problematik der vorangegangenen Entwicklung allein gelassen. Diese Vorgehensweise ist bei der Verwendung von LSI-Bausteinen nicht mehr zulässig, da diese Bauelemente zu vielfältig und komplex sind. Es ist die Verantwortlichkeit des Entwicklers, der die Funktion seines Systems am besten kennt, der Testabteilung die notwendigen Testvektoren zur Verfügung zu stellen.

Bild 1. Zeitlicher und logischer Ablauf von der Konzeption bis zur Funktionsprüfung programmierbarer Logik

Verifikation bipolarer PROMs

Bei bipolaren PROMs – in der Vergangenheit die einzigen programmierbaren Bauelemente, die in größeren Stückzahlen eingesetzt wurden – ist der Logiktest geradlinig.

```
PAL16H2                     PAL DESIGN SPECIFICATIONS
74LS153                     WILLY VOLDAN     15.01.85.
DUAL 4-ZU-1 MULTIPLEXER
MONOLITHIC MEMORIES GMBH, MÜNCHEN
1C0 1C1 1C2 1C3 A   B   2C0 2C1 2C2 GND
2C3 2G  NC  NC  2Y  1Y  NC  NC  1G  VCC

1Y =  /1G*/B*/A*1C0     ;SELECT 1C0
   +  /1G*/B* A*1C1     ;SELECT 1C1
   +  /1G* B*/A*1C2     ;SELECT 1C2
   +  /1G* B* A*1C3     ;SELECT 1C3

2Y =  /2G*/B*/A*2C0     ;SELECT 2C0
   +  /2G*/B* A*2C1     ;SELECT 2C1
   +  /2G* B*/A*2C2     ;SELECT 2C2
   +  /2G* B* A*2C3     ;SELECT 2C3

FUNCTION TABLE

B A 1C0 1C1 1C2 1C3 2C0 2C1 2C2 2C3 1G 2G 1Y 2Y
-----------------------------------------------
L L  L   X   X   X   L   X   X   X   L  L  L  L
L L  H   X   X   X   H   X   X   X   L  L  H  H
L H  X   L   X   X   X   L   X   X   L  L  L  L
L H  X   H   X   X   X   H   X   X   L  L  H  H
H L  X   X   L   X   X   X   L   X   L  L  L  L
H L  X   X   H   X   X   X   H   X   L  L  H  H
H H  X   X   X   L   X   X   X   L   L  L  L  L
H H  X   X   X   H   X   X   X   H   L  L  H  H
-----------------------------------------------
```

Bild 2. Eingabetext zur Assemblierung und Funktionstabelle zur Simulation mit PALASM

Ein PROM ist ein Lesespeicher, wobei Sicherungselemente die Speicherfunktion des Bauteils realisieren. Die Funktion eines PROMs ist das Lesen gespeicherter Informationen an einer bestimmten Adresse; d. h., um die Funktion eines solchen Bauelementes zu überprüfen, genügt es, nach der Programmierung die Sicherungsmatrix zu testen. Diese Verifikation der Sicherungsmatrix wird vom Programmiergerät automatisch durchgeführt. Arbeitet ein PROM nach der Programmierung einwandfrei, so ist der Baustein auch logisch in Ordnung.

Verifikation programmierbarer Logik

Bei der Verifikation der Sicherungsmatrix programmierbarer Logik erfolgt kein Logiktest des Bausteins. Programmierbare Logikbausteine werden zwar programmiert und verifiziert wie ein bipolares PROM, besitzen aber eine gegenüber PROMs unterschiedliche Konfiguration.

Der Eingangsdecoder hat ebenfalls Zugriff zur Sicherungsmatrix, wird jedoch nur während der Programmierung und Verifikation des Sicherungs-Arrays benutzt. Dafür sind die logischen Eingänge und Rückkoppeleingänge während der Programmierung und Verifikation der Matrix gesperrt und verlieren deshalb jegliche logische Beteiligung. Das bedeutet, daß eine Verifikation des Sicherungs-Arrays nur die korrekte Programmierung des Bausteins überprüft, nicht aber die korrekte logische Funktion.

Erstellung von Testvektoren mit PALASM

Eine einfache Möglichkeit zur Erstellung von Testvektoren bietet die Verwendung des im PALASM enthaltenen Simulationsprogrammes für PAL-Bausteine von Monolithic Memories.

Der Entwickler erstellt eine einfache Funktionstabelle, aus der die Software Testbedingungen ableitet, die sofort für die Überprüfung der logischen Funktion des PAL-Bausteins verwendet werden können. Das PAL-

```
PAL16H2                     PAL DESIGN SPECIFICATIONS
PATXXXX                     WILLY VOLDAN     15.01.85.
BEISPIEL FÜR SIMULATION UND FAULT-TESTING
MONOLITHIC MEMORIES GMBH, MÜNCHEN
*D2222*F0*G0*
L0000 11011110111110111011111111111111*
L0032 01111110111101111011111111111111*
L0064 11110110111110110111111111111111*
L0096 11111110011101110111111111111111*
L0256 11111111111110111011011111101111*
L0288 11111111111101111011111101101111*
L0320 11111111111110110111111111100111*
L0352 11111111111101110111111111101101*
V0001 0XXX000XXNX0XXLLXX0N*
V0002 1XXX001XXNX0XXHHXX0N*
V0003 X0XX10X0XNX0XXLLXX0N*
V0004 X1XX10X1XNX0XXHHXX0N*
V0005 XX0X01XX0NX0XXLLXX0N*
V0006 XX1X01XX1NX0XXHHXX0N*
V0007 XXX011XXXN00XXLLXX0N*
V0008 XXX111XXXN10XXHHXX0N*
793D
```

Bild 3. Programmierformat mit Sicherungs-Plot und Information über die Testvektoren nach JEDEC

Simulationsprogramm des PALASM vollführt zwei Prüfvorgänge:

1. Mit Hilfe der Funktionstabelle werden die logischen Gleichungen auf ihre Richtigkeit geprüft.
2. Die aus dieser Funktionstabelle während der Simulation erzeugten Testvektoren können sofort für die logische Verifikation des Bausteins herangezogen werden.

Bei sequentieller Logik lassen sich ganze Ablaufdiagramme mit Hilfe dieses Programmes simulieren bzw. nach der Programmierung testen (*Bild 1* und *2*).

Diese aus der Funktionstabelle des PAL-Programmes erzeugten Testvektoren werden nun mit der Programmierinformation in den Programmer geladen und ermöglichen ohne weiteren Aufwand einen 100 %igen Logiktest programmierbarer Logikbausteine.

JEDEC-Format

Das JEDEC-Format ist das internationale Standard-Format zur Übertragung von Programmierformaten und der Testinformation für programmierbare Logik. Es enthält eine Beschreibung zur Dokumentation, die dem ursprünglichen PALASM-Eingabetext entnommen ist, den Herstellercode, den Bauelemente-Code, die Programmierinformation und die Testvektoren. Es wird mit einer Checksumme zur Überprüfung der Übertragung der Daten abgeschlossen (*Bild 3*).

Testen in der Produktion

Oft steht in der Produktion kein Rechner zur Verfügung, der die oben erwähnten, strukturierten Testvektoren liefert. Dann empfiehlt sich die Möglichkeit, das vom PALASM generierte JEDEC-Format in einem EPROM abzuspeichern und es als „Master" für die weitere Duplizierung von PALs zu verwenden.

Für einfachste und ausschließlich kombinatorische Logik ist es auch möglich, die von verschiedenen Programmer-Herstellern angebotene Signatur-Analyse zu benutzen. Dabei wird eine pseudozufällige Sequenz von Testvektoren aus einem Schieberegister an das PAL angelegt und das Ergebnis an den Ausgängen wieder abgespeichert. Bei jedem nun zu duplizierenden PAL wird die gleiche Eingangssequenz angelegt und das Ergebnis mit dem bereits Abgespeicherten verglichen. Allerdings muß dabei ein Bauelement zur Verfügung stehen, von dem man sicher weiß, daß es in Ordnung ist.

Vorsicht: Dieser Test ist nur bei rein kombinatorischen PAL-Bauelementen erlaubt. PALs mit Rückkopplungen und Registerfunktionen sind für eine derartige Verifikation nicht geeignet, da auch vorhergehende Zustände berücksichtigt werden müssen, die diese Signatur-Analyse nicht kennen kann!

Willibald Voldan

Hinkebein – Taktverzögerungs-Schaltung mit PALs

Mit einem der flexiblen PAL-Bausteine kann man einfach ein Bauelement zur Impulsverzögerung herstellen, mit dem langsamere Peripheriebausteine angesteuert werden können. Die strukturierte Architektur der PALs ermöglicht eine Synchronisation der verlängerten Impulse mit der Taktfolge der CPU.

In diesem Beispiel wird die logische Verknüpfung zu einer solchen Schaltung im programmierbaren Array mit den Rückkoppel-Eingängen der Register gebildet.

Der „BOARD SELECT (B_SEL)"-Eingang dient zur Freigabe der gesamten Schaltung, die „FUNCTION SELECT (F_SEL0, F_SEL1 usw.)"-Eingänge steuern die jeweilige Impulsfolge. Wie aus dem Impulsdiagramm (*Bild 1*) hervorgeht, synchronisiert der Takt das Gesamtsystem.

Ein Systemtakt von 2 MHz erzeugt z. B. eine Zykluszeit von 0,5 µs. Mit der nachfolgenden Impulsverzögerungsschaltung im PAL-Baustein können nun verlängerte Impulse mit einem Vielfachen von 0,5 µs, also 1,0, 1,5, 2,0, 2,5 µs und so weiter, erzeugt werden.

Die Periodenlänge ist beliebig über die Programmierung des PALs zu variieren. Die logischen Gleichungen (*Bild 2*) dienen zur Programmierung des PAL-Bausteins zur Erzeugung des gewünschten Impulsdiagramms.

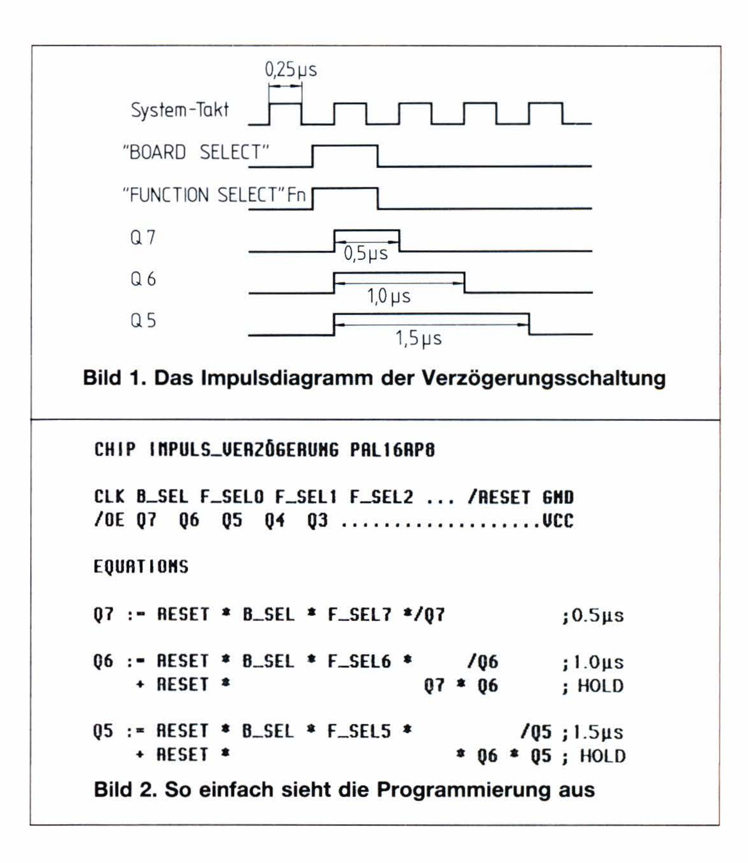


Bild 1. Das Impulsdiagramm der Verzögerungsschaltung

```
CHIP IMPULS_VERZÖGERUNG PAL16RP8

CLK B_SEL F_SEL0 F_SEL1 F_SEL2 ... /RESET GND
/OE Q7  Q6  Q5  Q4  Q3 ...................VCC

EQUATIONS

Q7 := RESET * B_SEL * F_SEL7 */Q7              ;0.5µs

Q6 := RESET * B_SEL * F_SEL6 *       /Q6       ;1.0µs
    + RESET *                    Q7 * Q6       ; HOLD

Q5 := RESET * B_SEL * F_SEL5 *            /Q5 ;1.5µs
    + RESET *                       * Q6 * Q5 ; HOLD
```

Bild 2. So einfach sieht die Programmierung aus

Magsoodul Mannan

PALs richtig getestet

Wegen ihrer relativ leicht durchschaubaren inneren Struktur eignen sich PALs vorzüglich für den softwaregesteuerten Test. Diese Einfachheit darf dennoch nicht dazu verleiten, die Prüfung als völlig unproblematisch zu betrachten. Das hier beschriebene Softwarepaket berücksichtigt die häufigsten Fehler und ist zudem so konzipiert, daß meist keine Unterstützung durch einen Spezialisten nötig ist.

Entwurfs- und Test-Prozeß

Bild 1 zeigt schematisch den Entwurfs- und Test-Zyklus für PAL-Bausteine. Der erste Schritt ist die Generierung der Logikgleichungen und der Funktionstabelle. Logikgleichungen lassen sich z. B. mit Hilfe des Softwarepaketes „PLAN" (Programmable Logic Analysis) von National Semiconductor aufstellen. PLAN ist eine auf einem PC lauffähige Software, mit der sich alle Boole'schen Gleichungen generieren lassen.

Bild 1. Entwurf und Testzyklus für einen PAL-Baustein

Bild 2. ▷ **Automatische Testgenerierung**

Nachdem die logischen Gleichungen sowie die Funktionstabelle existieren, sind die Bit-Muster oder Durchschmelz-Anweisungen („Fuse Plots") zu generieren, die bestimmen, welche Verbindungen aufgetrennt werden müssen. Nun läßt sich ein Vergleich zwischen den Logikgleichungen und der Funktionstabelle vornehmen. Zur Erzeugung des Bitmusters eignen sich verschiedene Programme, zum Beispiel PALASM (MMI) oder PLAN (National Semiconductor).

Der nächste Schritt ist das Laden des Bitmusters in das PAL-Programmiergerät zur Programmierung und Verifizierung der Verbindungsmatrix.

Danach erfolgt der Funktionstest des PAL-Bausteins. Für diesen Prozeß sind Testvektoren erforderlich. Ausgehend von den Logikgleichungen, die im ersten Schritt generiert wurden, wird die ganze Schaltung simuliert.

Schließlich werden automatisch die Testmuster erzeugt, mit denen eine Fehlersimulation möglich ist.

Bild 1 gibt einen Überblick über den gesamten Prozeß sowie die Stufen, die ein Bauelement während des Entwicklungszyklus durchläuft. Obwohl der Test-Generierungsprozeß vom gleichen Ausgangspunkt startet, folgt er einem anderen Weg und gibt dem Testingenieur deshalb von der Entwicklung unbeeinflußte, zuverlässige Testdaten für diese Bauelemente.

Test-Generierungsprozeß

Bild 2 gibt einen ausführlichen Überblick über den Test-Generierungsprozeß. Dieser beginnt mit den Booleschen Gleichungen und durchläuft einen Vorverarbeitungsschritt zur Prüfung des Formates. Anschließend wird eine Netzliste mit Hilfe eines Umsetzungsprogramms generiert. Diese Liste enthält die exakte Schaltungsbeschreibung und benutzt bestimmte System-Pri-

mitive, die in einer Bibliothek enthalten sind. Für PAL-Bausteine sind das nur wenige Primitive, die aus UND- und ODER-Gattern, Puffern und Latches bestehen.

Bevor sich die Netzliste simulieren läßt, sind noch zusätzliche Eingabedateien zu erzeugen. Diese beziehen sich auf Strobe- und Zeit-Informationen. Darüber hinaus muß die Schaltung in einen definierten Anfangszustand gebracht werden (Initialisierung). Dies ist besonders wichtig für sequentielle Schaltungen, wenn in den verwendeten PAL-Bausteinen die Flipflops keine Preset- oder Clear-Leitungen besitzen. Die Logikgleichungen werden geprüft; handelt es sich um ein Bauelement mit Register, so wird die Initialisierungssequenz generiert.

Als nächster Schritt erfolgt die Herstellung der Testmuster. Verschiedene Algorithmen führen in unterschiedlichen Situationen zu verschiedenen Resultaten.

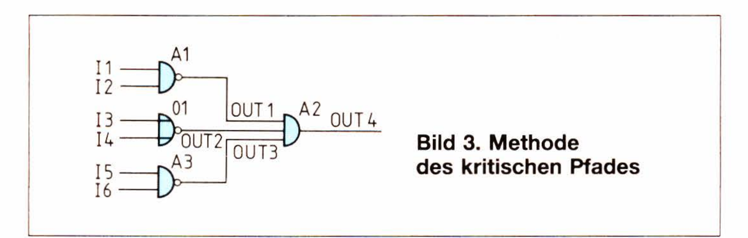


Bild 3. Methode des kritischen Pfades

Am weitesten verbreitet sind der sogenannte „D-Algorithmus" und die Methode des „kritischen Pfades". Der D-Algorithmus ist außerordentlich effizient, allerdings nicht einfach zu handhaben. PLAN verwendet die Methode des kritischen Pfades zur Generierung des Tests. Ein Primärausgang wird dabei ausgewählt, zu 0 oder zu 1 angenommen und dann rückwärts verfolgt, bis ein Primäreingang erreicht ist.

Bild 3 zeigt, wie diese Methode funktioniert. Hier sind zwei NAND-Gatter und ein NOR-Gatter miteinander an einem UND-Gatter verbunden, um ein UND-Gatter zu formen. Zur Generierung des Tests wird zunächst Ausgang „OUT 4" auf Null gesetzt. Als nächstes wird zur Definition der kritischen Eingänge von UND-Gatter A2 OUT 1 = 0, OUT 2 = 1 und OUT 3 = 1 gesetzt oder OUT 1 = 1, OUT 2 = 0 und OUT 3 = 1 oder jede andere Kombination, solange der Ausgang OUT 4 auf Null bleibt. Bei der ersten Möglichkeit ergibt sich im Falle von OUT 1 = 0, daß I 1 = I 2 = 1 und beide Eingänge von A1 kritisch sind. Der Zustand OUT 2 = 1 erfordert I 3 = I 4 = 0, und OUT 3 = 1 erfordert I 5 = I 6 = 0. Bei dieser Version wurde damit das Testmuster 110000 generiert. In gleicher Weise lassen sich für die anderen Möglichkeiten Testmuster generieren.

Bei einigen redundanten Schaltungen ist die Analyse nicht vom Ausgang ausgehend durchführbar, da sich dabei kein eindeutiges Testmuster ergibt. In diesem Fall gibt der Testgenerator entsprechende Meldungen an den Benutzer aus.

Der nächste Schritt ist die Fehlersimulation. Es werden nur einzelne Haftfehler (stuck at 1/0) simuliert, obwohl eine Mehrfachfehlersimulation ebenfalls möglich ist.

Das endgültige Testmuster aus dem Fehlersimulator entspricht keinem der üblichen Testerformate und muß daher formatiert werden. Mit Hilfe von Software lassen sich die Testvektoren in die üblichen Testerformate umsetzen (Sentry, Megatest oder die JEDEC-Formate). Die Ausgangscodes dieser Umsetzungsprogramme lassen sich dann auf Band überspielen, z. B. mit Hilfe des Sentry-Megatesters kompilieren und für den Funktionstest benutzen.

Der gesamte Weg von den Logikgleichungen zu testerformatierten Daten wird mit Hilfe von Menüs gesteuert und läuft vollständig automatisch ab. Es ist weder ein besonders ausgebildeter Ingenieur noch ein Softwareexperte erforderlich. Jeder, der sich mit Computerterminals auskennt, ist in der Lage, das Programm zu benutzen. Nur wenn bestimmte komplizierte Schaltungen getestet werden sollen, ist die Unterstützung durch qualifiziertes Personal erforderlich.

Erweiterte Softwareunterstützung

Nachdem ein bestimmtes PAL-Bauelement ausgewählt ist, muß das Verbindungsmuster entworfen werden. Dies läßt sich entweder manuell oder mit Hilfe bestimmter Softwarepakete durchführen, die kommerziell erhältlich sind. Manuelle Codierung, die bei einfachen Schaltungsentwürfen durchaus realisierbar ist, steckt voller Fehlerquellen, so daß sich dieses Verfahren bei größeren PAL-Volumen verbietet. Aus diesem Grund wurde auch in dem in Bild 2 gezeigten Beispiel ein Softwarepaket zur Generierung der Bit-Tabellen herangezogen. Das Ergebnis ist dann ein Band, das man als Eingabemedium für die meisten Programmiergeräte verwenden kann.

Bild 4 zeigt typische Logikgleichungen und einen Teil der Bit-Tabelle, die für diesen Testfall generiert wurden. Die Nullen in der Bit-Tabelle stellen die vom Programmiergerät durchgeschmolzenen Verbindungen und die Einsen die intakten Verbindungen dar. Die letzte Spalte enthält eine Prüfsumme der betreffenden Zeilen. Die Bitmuster sind damit ein wertvolles Hilfsmittel beim Test und Entwurf sowie der Programmierung von PAL-Bausteinen.

Probleme

Die Software zur Testerstellung, die hier beschrieben wurde, eignet sich sehr gut für sequentielle Schaltungen und die meisten kombinatorischen Logiken. Wenn kombinatorische Schaltungen jedoch zu viele Rückführungsleitungen haben, insbesondere direkte Rückkopplungen vom Primärausgang zum Primäreingang, kann der Testgenerator leicht ungeeignete Testvektoren generieren. Diese asynchronen Rückführungen machen die Schaltung instabil und führen daher zu sehr unbrauchbaren Resultaten.

Die hier vorgestellte Software diente bereits zum intensiven Test Tausender Bausteine und hat, abgesehen

```
PAL TEST DEVICE
THE FIRST FOUR LINES ARE FOR COMMENTS ONLY AND THE 5TH AND
6TH LINES ARE FOR INPUT/OUTPUT LIST. SLASH (/) MEANS COMPLEMENT.
DATE 10th OF JULY, 1984
S1 S2 /S3 S4 S5 S6 S7 S8 S9 GND
/S11 S12 S13 /S14 /S15 /S16 /S17 /S18 S19 VCC
S15 = /S2 * /S5 * /S6 * /S8 *  S4  * S3 * /S11
S18 =  S2 *  S5 *  S6 * /S1 * /S13 * S3 * /S11
S14 =  S2 *  S5 * /S6 *  S1 * /S13 * S3 * /S11
S16 =  S2 *  S5 * /S6 * /S8 *  S1  * S13*  S3 * S11 * /S7 +
       S2 *  S5 * /S6 *  S8 * /S9  * S1 * /S13*  S3 * S11 * S7
/S19= /S2 * /S5 * /S6 * /S8 *  S4  * S3 * /S11 +
       S2 *  S5 * /S13*  S3 * /S11 + /S3 + S11
/S12= /S18* /S14* /S16* /S17

        A 0 0000 0000 0000 0000 0000 0000 0000 0000 2  2
        A 1 0100 0100 1000 0100 0100 0000 0100 0010 6 13
        A 2 1000 0100 0000 1000 0000 0000 0001 0010 3  8
        A 3 0000 1000 0000 0000 0000 0000 0000 0000 4  5
        A 4 0000 0000 0000 0000 0000 0000 0000 0001 3  4
        A 5 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A 6 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A 7 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A 8 0000 0000 0000 0000 0000 0000 0000 0000 2  2
        A 9 1001 0100 0000 1000 1000 0000 0001 0010 6 13
        A10 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A11 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A12 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A13 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A14 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A15 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A16 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A17 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A18 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A19 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A20 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A21 1111 1111 1111 1111 1111 1111 1111 1111 5 37
        A22 1111 1111 1111 1111 1111 1111 1111 1111 5 37
```

Bild 4. Logische Eingangsgleichungen und Teil einer Bit-Tabelle für den Testvorgang

von den Problemen mit asynchronen Rückführungen, bei kombinatorischen Schaltungen sehr gute Resultate gezeigt. Asynchron rückgekoppelte Schaltungen müssen analysiert werden, bevor sie simuliert werden. Wenn sich dabei herausstellt, daß ein Stabilitätsproblem vorliegt, können sie unter vollständig anderen Bedingungen simuliert werden. Solche Spezialsimulatoren werden in der nächsten Generation der Testalgorithmen vorhanden sein.

Erfahrungen

Bei der Konzipierung der Software PLAN wurde insbesondere darauf geachtet, daß die Arbeit des Testingenieurs erleichtert wird. Hier müssen keine Wahrheitstabellen manuell erstellt werden; die Testvektoren ergeben sich einfach beim Ausfüllen der Menüs, die auf dem Bildschirm zu sehen sind. Die Software eignet sich sowohl für interaktiven als auch Batch-Betrieb und gestattet dem Testingenieur die Überwachung des Testgenerierungsprozesses.

PAL-Bausteine sind sehr gut für solche automatisierten Prozesse geeignet, weil sie entsprechende Strukturen aufweisen. Darüber hinaus wird die Softwareunterstützung für PAL-Bausteine kontinuierlich ausgebaut; auch die heute noch bestehenden Probleme dürften also in Zukunft zu lösen sein.

Der Autor ist Mitarbeiter von National Semiconductor in Santa Clara, Kalifornien.



Programmieren Sie jetzt Ihre Zukunft

– wir beherrschen alle Technologien.



Kontron Elektronik, der führende europäische Hersteller digitaler Meßtechnik, liefert das breiteste Spektrum an Systemen auch für Ihre Entwicklungsaufgaben:
Programmiergeräte, Logikanalysatoren, In-Circuit-Emulatoren, CAD- und Entwicklungssysteme.

KONTRON's neues **U**niversal **P**rogrammier **M**odul **UPM** ist für alle Bauteilvarianten programmierbarer Logic, MOS und Bipolar ausgerüstet.
Dieses einzigartige Leistungsspektrum basiert auf der Anwendung neuester Gate Array- und Hybrid-Technologie.

Selbstverständlich ist auch dieses Modul zu allen unseren portablen PROM Programmern MPP-80S und Tischgeräten EPP-80 kompatibel.

Testen Sie die Leistungsfähigkeit unserer zukunftsorientierten Programmier-Systeme.

KONTRON ELEKTRONIK GRUPPE

KONTRON MESSTECHNIK

8057 Eching b. München
Oskar-von-Miller-Str. 1
Telefon (081 65) 77-0
Telex 526 719
Telefax (081 65) 77-385

TECHNISCHE BÜROS

8500 Nürnberg 20
Rennweg 60/62
Tel. (09 11) 53 33 06
Telex 6 26 391

7000 Stuttgart 30
Maybachstraße 39a
Tel. (07 11) 89 17-0
Telex 7 23 061

6000 Frankfurt 70
Kennedy-Allee 34
Tel. (069) 63 17-0
Telex 4 14 881

4000 Düsseldorf 1
Ronsdorfer Str. 145
Tel. (02 11) 73 61-0
Telex 8 582 675

3000 Hannover 81
Hermann-Guthe-Str. 3
Tel. (05 11) 83 90 51-57
Telex 9 23 729

2000 Hamburg 70
Königsreihe 2
Tel. (0 40) 6 82 95-0
Telex 2 11 998

1000 Berlin 41
Albrechtstraße 34
Tel. (0 30) 792 30 31-3
Telex 1 85 484



Otto Schöfbeck

Das JEDEC-Format in Theorie und Praxis

Die Schritte, die notwendig sind, um von einer Schaltung zu einem programmierten PAL zu kommen, sind oft sehr mühsam. Man verwendet häufig Werkzeuge verschiedener Hersteller, deren Zusammenschaltung meist nicht auf Anhieb läuft. Dieser Beitrag versucht, dem Anwender die manchmal geradezu als Mysterium angesehenen Übertragungsprotokolle ein wenig nahezubringen und auf mögliche Fehlerquellen einzugehen, die beim Zusammenschalten eines PAL-Assemblers mit einem Programmiergerät auftreten. Alle Beispiele beziehen sich auf den PALASM von MMI und ein Programmiergerät von Kontron mit dem UPM-Modul. Das Bindeglied zwischen einem Computer mit einem geeigneten PAL-Assembler und einem Programmiergerät ist meist eine serielle Schnittsteile. Mögliche Protokolle mit ihren Vor- und Nachteilen sowie Fehlerquellen und ihre Beseitigung sollen in diesem Teil beschrieben werden.

Als die ersten PALs aufkamen, sollte ihre Programmierung auf bereits existierenden PROM-Programmern möglich sein. Deshalb wurde auch die Art der Programmierung so ausgelegt, daß man adreßbezogen die Daten in ein Programmiergerät übertrug und wie bei PROMs Sicherung für Sicherung programmierte.

Das Programmierarray der 20-Pin-PALs von MMI entspricht einem PROM mit der Größe 512 × 4 Bit, es enthält also 2048 Sicherungen. *Bild 1* zeigt das Innenleben eines PAL12H6. Die senkrechten Linien stellen die Eingänge dar, die waagrechten Linien sind die "Produkt-Terme" (PT). An jedem möglichen Kreuzungspunkt (maximal 2048) in dem Diagramm befindet sich eine Sicherung, die über geeignete Beschaltung von außen adressierbar und lesbar ist. Der Zusammenhang zwischen den einzelnen Sicherungen und den Daten im Programmiergerät ergibt sich folgendermaßen:

Faßt man acht PTs zu einer Gruppe zusammen, so erhält man 256 Sicherungen pro Gruppe. Vier solcher Gruppen sind das halbe PAL-Array und belegen 256 x 4 Bits. Der ersten Gruppe (0...7) wurde Bit 0, der zweiten Gruppe (8...15) Bit 1 u.s.w. zugeteilt. Für die zweite Hälfte gilt exakt dieselbe Vereinbarung. Zum Programmieren eines PALs muß man die vom PALASM generierte PROM-Matrix in ein Programmiergerät übertragen.

Natürlich ist der sehr aufwendige Weg über handcodierte Tabellen und das manuelle Setzen der einzelnen Speicherstellen prinzipiell möglich. Dies ist jedoch nicht nur zeitraubend, sondern auch fehlerträchtig.

Für die Übertragung erzeugt der PAL-Assembler von MMI die Darstellung in einem HEX-Format, das dann entweder direkt an eine serielle Schnittstelle kopiert oder als File auf einer Diskette abgelegt werden kann. Dabei wird jeder mögliche 4-Bit-Wert in ein ASCII-Zeichen (0...9 und A...F) umgewandelt und durch

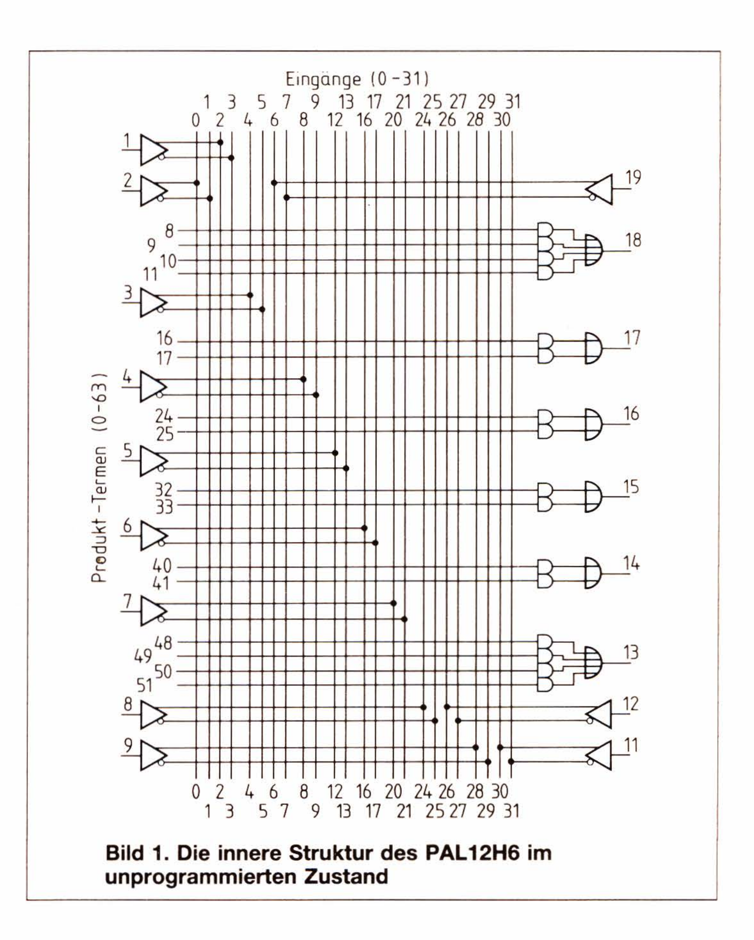
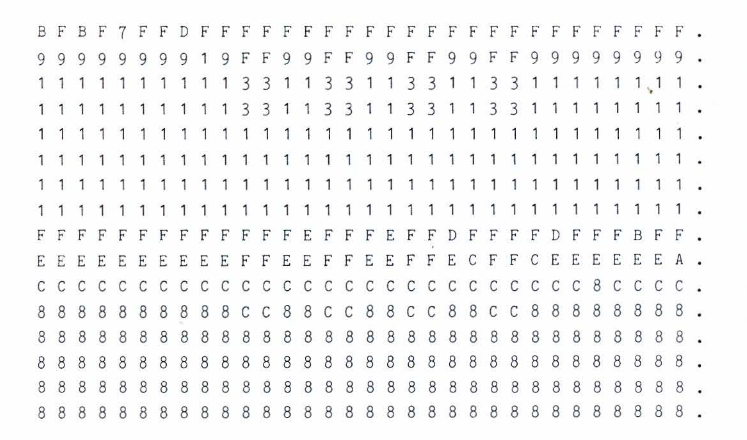


Bild 1. Die innere Struktur des PAL12H6 im unprogrammierten Zustand

Spaces bzw. eine Zeilenschaltung (CRLF) in eine druckbare Form gebracht. Nachstehend ist ein Ausdruck für ein PAL mit 20 Anschlüssen dargestellt. Die Daten gehören zu einem Musterbeispiel, auf das im folgenden noch näher eingegangen wird.



```
B F B F 7 F F D F F F F F F F F F F F F F F F F F F F F F F F F .
9 9 9 9 9 9 9 9 1 9 F F 9 9 F F 9 9 F F 9 9 F F 9 9 9 9 9 9 9 9 .
1 1 1 1 1 1 1 1 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 1 1 1 1 1 1 .
1 1 1 1 1 1 1 1 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 1 1 1 1 1 1 1 1 .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 .
F F F F F F F F F F F F F E F F F E F F D F F F F D F F F B F F .
E E E E E E E E E E F F E E F F E E F F E C F F C E E E E E E A .
C C C C C C C C C C C C C C C C C C C C C C C C C C C 8 C C C C .
8 8 8 8 8 8 8 8 8 8 C C 8 8 C C 8 8 C C 8 8 C C 8 8 8 8 8 8 8 8 .
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 .
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 .
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 .
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 .
```

Die Einfachheit des Protokolls läßt eine sichere Fehlererkennung nicht zu, da es keine Prüfsumme enthält. Lediglich die Anzahl der HEX-Zeichen und der zu erwartende Zeichenumfang sind überprüfbar.

Mit dem Übergang auf komplexere Bausteine erhöht sich natürlich auch die Anzahl der programmierbaren Sicherungen. Das Übertragungsprotokoll für die 20-Pin-PALs kann man hier nicht mehr übernehmen, da ein Teil der Sicherungsinformation damit nicht übertragen werden kann. Die interne Datenstruktur von 24-Pin-PALs erfordert es, für insgesamt 40 Eingänge und 80 PTs ein PROM-Array mit insgesamt 40 x 80 = 3200 Sicherungen zu übertragen.

MMI-PALs benutzen eine Arraygröße von 640 x 5. Man erkennt, daß das oben gezeigte MMI-Hex-Format für 20-Pin-PALs nun nicht mehr zur Übertragung der Daten hergenommen werden kann. Durch Erweiterung der Anzahl (640) und des Zeichenumfangs (00..1F, entsprechend 5 Bit) läßt sich jedoch das gesamte Programmierarray erfassen. Ein Auszug ist im folgenden dargestellt.

```
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 .
19 18 19 19 19 18 19 19 19 18 19 19 01 19 19 19 01 19 01 19 .

11 11 11 11 11 10 01 11 11 10 01 11 11 10 11 11 11 10 11 11 .
11 10 11 11 11 11 11 11 11 10 11 11 01 11 11 11 01 11 01 11 .
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .

1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F .
1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F .
1F 1F 1F 1F 17 1F 0F 1F 1B 1F 0F 1F 1D 1F 1F 1F 1E 1F 1F 1F .
1F 1F 1F 1F 10 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F .
0A 0C 06 0C 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E .
00 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E .
```

Diese beiden Beispiele verdeutlichen, daß steigende Arraygröße auch ein verändertes Datenprotokoll verlangt. Da andere PAL-Hersteller mitunter auch andere Programmier-Arrays verwenden, liegt es nahe, eine Übertragungsart zu finden, die nicht an irgendein bestimmtes Array gebunden ist, eine Fehlererkennung erlaubt und auch Zusatzinformationen beinhalten kann. Hat man z. B. eine Möglichkeit, den PAL-Typ in codierter Form zu übertragen, so kann sich ein „intelligentes" Programmiergerät nach der Datenübertragung automatisch auf den entsprechenden Typ einstellen. Eine solche Art der Übertragung wurde 1980 vom JEDEC-Komitee (Joint Electronic Devices Engineering Council) erstellt. Sie ist als JEDEC-Datenprotokoll bekannt geworden. Meist erzeugt ein PAL-Assembler einen ASCII-File auf einem Speichermedium (Disk), der sich dann über eine serielle Schnittstelle an ein Programmiergerät übertragen läßt.

Fassen wir alle Informationen zusammen, die notwendig sind, um Daten für ein PAL in den Datenspeicher eines Programmiergerätes zu übertragen:

- PAL-Typ
- Hersteller
- Sicherungsinformation
- Prüfsumme

Damit ein Programmiergerät diese Information verstehen und sinnvoll auswerten kann, müssen gewisse Regeln beachtet werden und alle Informationen formatiert sein. Mit „formatiert" bezeichnet man hier die Einteilung der Daten in Felder, wobei jedes Feld entsprechend der Information, die es enthält, durch ein Zeichen für Feldanfang und -ende markiert ist.

Das Steuerzeichen STX (02H) aktiviert den JEDEC-Empfänger im Programmiergerät. Alle vorher empfangenen Zeichen, die z. B. die Beschreibung des Bausteins im Klartext enthalten, werden nicht ausgewertet. Es wird eine Prüfsummenbildung gestartet, die sämtliche nun ankommenden Zeichen mitzählt und aufsummiert. Zur Kennzeichnung eines Feldes dienen verschiedene ASCII-Zeichen, die sysmbolisch für den Inhalt des Feldes stehen. Will man nun dem Programmiergerät mitteilen, daß die folgenden Daten sich auf ein PAL12H6 beziehen, so erfolgt das durch folgende Eingabezeile:

D2219∗(CRLF)

Der Buchstabe „D" steht für Device Code, die Ziffer 22 ist der Herstellercode für MMI und 19 bezeichnet den Typ PAL12H6. Das nachfolgende Zeichen „∗" beendet dieses Feld. Steuerzeichen, wie CR, LF und Zwischenräume, sind nur für einen Ausdruck notwendig; sie beeinflussen zwar die Übertragungs-Prüfsumme, werden aber sonst nicht vom Empfänger beachtet, da sie keine Information enthalten. Erst nach einem Feldende-Zeichen kann ein neues Feld erkannt werden.

Die Feldkennzeichnung für die Sicherungen ist ein „L" für Link, dem eine 4stellige Dezimalzahl folgt. Damit läßt sich jede Sicherung in einem PAL selektiv setzen. Der eigentliche Zustand der jeweiligen Sicherung wird dann in diesem Feld durch eine 0 oder 1 (in ASCII) bestimmt, wobei „0" für eine intakte und „i" für eine programmierte Sicherung steht. Mit dem Feld

L0056 0110∗(CRLF)

bleibt z. B. die Sicherung 56 unverändert, d. h. unpro-

grammiert, während Sicherungen 57 und 58 programmiert werden. Sicherung 59 bleibt unprogrammmiert. Da dieses Feld mit einem Feldendezeichen abgeschlossen ist, kann für weitere Sicherungsinformationen ein neues Feld mit einer neuen Sicherungsnummer folgen. Ein Abschluß der Zeile mit (CRLF) bewirkt, daß die nächstfolgende Sicherung gesetzt wird.

Durch die Folge

```
L0056 0110  (CRLF)
      1100* (CRLF)
```

werden die Sicherungen 60 und 61 programmiert, 62 und 63 bleiben intakt.

Ein weiteres Feld ist erlaubt, um alle Sicherungen gleichzeitig in einen bestimmten Zustand zu bringen:

F0*

oder

F1*

Das „F" steht hier für „Default setzen", das nachfolgende ASCII-Zeichen „0" setzt alle Sicherungen in den unprogrammierten Zustand, 1 setzt alle Sicherungen in den programmierten Zustand.

Das Ende der Übertragung wird mit dem Kontrollzeichen „ETX" markiert. Danach liest der Empfänger noch vier Zeichen, die die vom Sender berechnete Übertragungs-Prüfsumme beinhalten. Damit ist die Übertragung abgeschlossen. Durch Vergleich mit der übertragenen und der im Empfänger berechneten Summe kann man Übertragungsfehler erkennen.

Es wäre durchaus denkbar, mit einem Texteditor in eine JEDEC-Datei einzugreifen, wenn der PAL-Assembler z. B. keinen Device-Code erzeugt und man diesen manuell einfügen will. Das Verändern eines Zeichens wirkt sich allerdings auch auf die Prüfsumme aus; beim Übertragen würde der Empfänger also einen Fehler erkennen. Diesen Summenvergleich kann man aber auch umgehen, indem man die Prüfsumme vom Sender auf 0000 setzt; der Empfänger ignoriert dann den Vergleich.

Ein einfaches Beispiel aus dem MMI-Datenbuch soll nun zeigen, wie die einzelnen Schritte aussehen. Hierbei werden mit einem PAL12H6 insgesamt sechs verschiedene Gatter realisiert. *Bild 2* stellt die geforderte Schaltung dar.

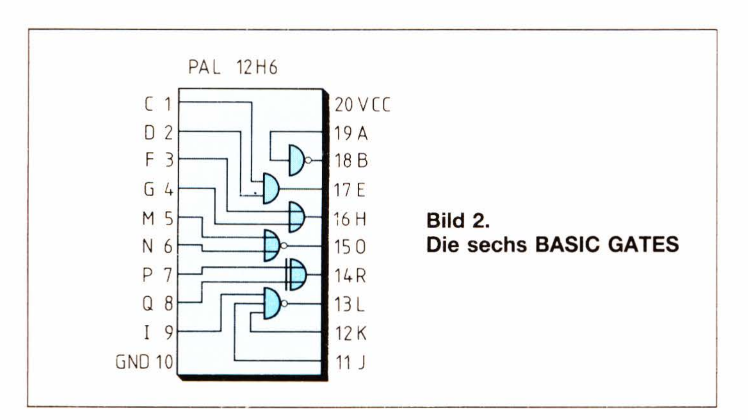


Bild 2.
Die sechs BASIC GATES

Zunächst sind für die Schaltung die Gleichungen aufzustellen und mit einem geeigneten PAL-Assembler zu übersetzen:

```
PAL12H6                      PAL DESIGN SPECIFICATION
P7000                          VINCENT COLI 03/12/82
BASIC GATES
MMI SUNNYVALE, CALIFORNIA
C D F G M N P Q I GND J K L R O H E B A VCC

B = /A                  ;INVERTER
E = C*D                 ;UND
H = F + G               ;ODER
L = /I + /J + /K        ;NAND
O = /M*/N               ;NOR
R = P*/Q + /P*Q         ;EXKLUSIV-ODER
```

Für dieses Beispiel liefert der PAL-Assembler das vorher schon gezeigte HEX-Format. Das JEDEC-Format würde dagegen wie folgt aussehen:

```
PAL12H6                      PAL DESIGN SPECIFICATION
P7000                          VINCENT COLI 03/12/82
BASIC GATES
MMI SUNNYVALE, CALIFORNIA

(STX)*D2219*F0*
L0000 111111011111111111111111*
L0096 010111111111111111111111*
L0144 111101111111111111111111*
L0168 111111101111111111111111*
L0192 111111111110101111111111*
L0240 111111111111110110111111*
L0264 111111111111111001111111*
L0288 111111111111111111111011*
L0312 111111111111111111111110*
L0336 111111111111111111101111*
(ETX) 0000
```

Die ersten vier Zeilen enthalten Informationen, die nicht zum Programmieren des Bausteins notwendig sind. Im D-Feld wird das PAL12H6 festgelegt, mit dem F-Feld und 0 werden dann alle programmierbaren Sicherungen unprogrammiert gesetzt. Das erste L-Feld setzt insgesamt 24 Sicherungen, davon soll nur Sicherung 7 unprogrammiert sein (1 entspricht programmiert, 0 entspricht unprogrammiert). Die Zeile ist mit einem "*" abgeschlossen, womit sich ein neues Feld übertragen läßt. Die nächste Zeile fährt bei Sicherung 96 fort; die Sicherungen 24 bis 95 werden also überspungen. Nach dem Übertragen der nächsten Zeile sind die Sicherungen 96 und 98 unprogrammiert, während 97 und 99 bis 119 programmiert werden.

Man erkennt hier den Vorteil, den das JEDEC-Format gegenüber allen anderen Übertragungsmethoden hat:

Es brauchen nur die Sicherungen angegeben zu werden, die zu programmieren sind, alle anderen braucht man nicht zu übertragen. Das Beispiel zeigt, daß in 10 L-Feldern insgesamt 240 Sicherungen übertragen werden,

während ein PAL12H6 insgesamt 384 programmierbare Sicherungen besitzt.

Im folgenden wurden die nicht gesendeten L-Felder ergänzt. Diese enthalten aber keine zusätzliche Information und liefern ein exakt identisches Ergebnis.

```
PAL12H6                    PAL DESIGN SPECIFICATION
P7000                         VINCENT COLI 03/12/82
BASIC GATES
MMI SUNNYVALE, CALIFORNIA

(STX)*D219*F0*
L0000 111111101111111111111111*
L0024 000000000000000000000000*
L0048 000000000000000000000000*
L0072 000000000000000000000000*
L0096 010111111111111111111111*
L0120 000000000000000000000000*
L0144 111101111111111111111111*
L0168 111111110111111111111111*
L0192 111111111110101111111111*
L0216 000000000000000000000000*
L0240 111111111111110110111111*
L0264 111111111111111001111111*
L0288 111111111111111111111011*
L0312 111111111111111111111110*
L0336 111111111111111111101111*
L0360 000000000000000000000000*
(ETX) 0000
```

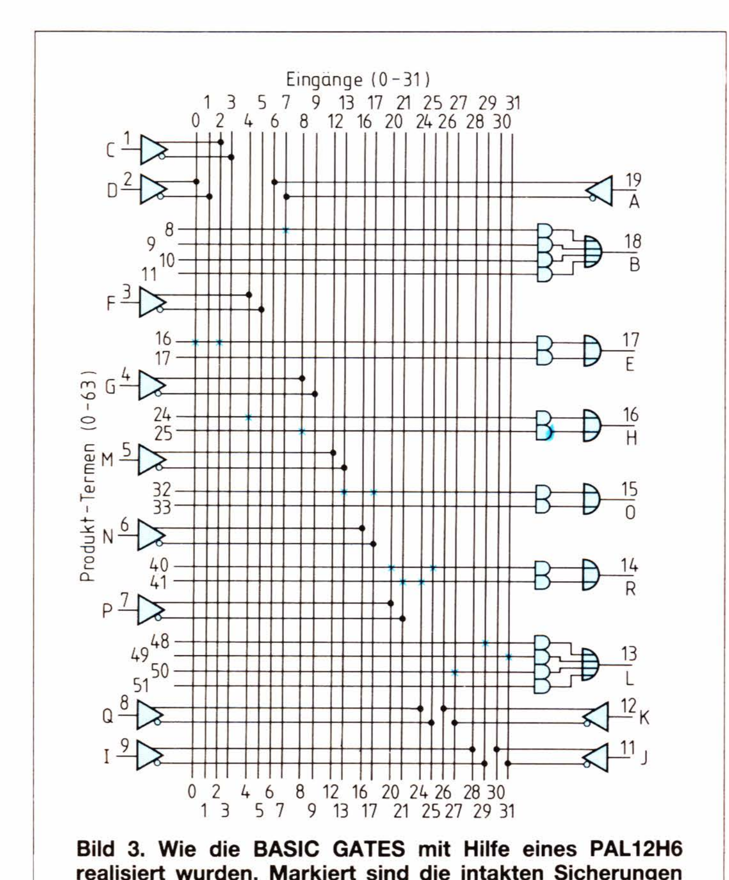


Bild 3. Wie die BASIC GATES mit Hilfe eines PAL12H6 realisiert wurden. Markiert sind die intakten Sicherungen

Auch der folgende Ausdruck enthält dieselbe Informationen. Das L-Feld wurde nur einmal gesetzt und erst nach der Übertragung aller Sicherungen mit „*“ abgeschlossen. Zum Beweis für die Flexibilität des JEDEC-Datenprotokolls werden im folgenden Beispiel zuerst alle Sicherungsinformationen übertragen, erst danach das Device-Feld. Die Kommentare nach jeder Sicherungszeile sollen nur angeben, zu welcher Zeile des Schaltbildes in *Bild 3* diese Sicherungsinformationen gehören. Das gilt auch für die Ziffern vor der ersten Sicherungsinformation; sie entsprechen den Bezeichnungen der Eingänge.

```
PAL12H6                    PAL DESIGN SPECIFICATION
P7000                         VINCENT COLI 03/12/82
BASIC GATES
MMI SUNNYVALE, CALIFORNIA

(STX)*
L0000
           11 11 22 22222233
0123456789 23 67 01 45678901

1111111011 11 11 11 11111111  8
0000000000 00 00 00 00000000  9
0000000000 00 00 00 00000000 10
0000000000 00 00 00 00000000 11

0101111111 11 11 11 11111111 16
0000000000 00 00 00 00000000 17

1111011111 11 11 11 11111111 24
1111111101 11 11 11 11111111 25

1111111111 10 10 11 11111111 32
0000000000 00 00 00 00000000 33

1111111111 11 11 01 10111111 40
1111111111 11 11 10 01111111 41

1111111111 11 11 11 11111011 48
1111111111 11 11 11 11111110 49
1111111111 11 11 11 11101111 50
0000000000 00 00 00 00000000 51
*
D2219*F0*(ETX)0000
```

Beim Betrachten des oben dargestellten JEDEC-Files erkennt man, daß für jeden Kreuzungspunkt, hinter dem eine programmierbare Sicherung ist, ein Sicherungszustand vorhanden ist. Dieser läßt sich direkt in das logische Schaltbild übertragen. Da nur eine 0 eine intakte Sicherung darstellt, ergibt sich ein fertig programmiertes PAL entsprechend Bild 3.

Berechnung der JEDEC-Sicherungsnummer

Die Einhaltung einer bestimmten Reihenfolgen (z. B. zuerst Device-Feld, dann Default-Feld und schließlich L-Feld) sollte bei einem „intelligenten“ JEDEC-Empfänger in einem Programmiergerät nicht vorgeschrieben sein.

Ein sehr wichtiger Punkt ist hier allerdings zu beachten. Das JEDEC-Datenprotokoll überträgt im L-Feld nur Sicherungsnummern und die dazugehörigen Siche-

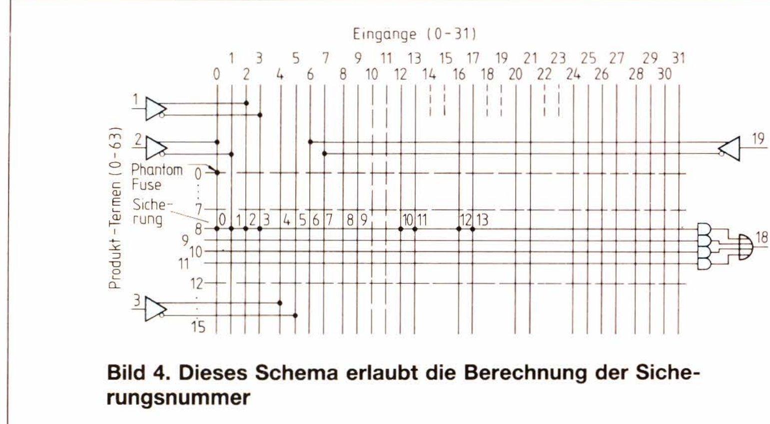


Bild 4. Dieses Schema erlaubt die Berechnung der Sicherungsnummer

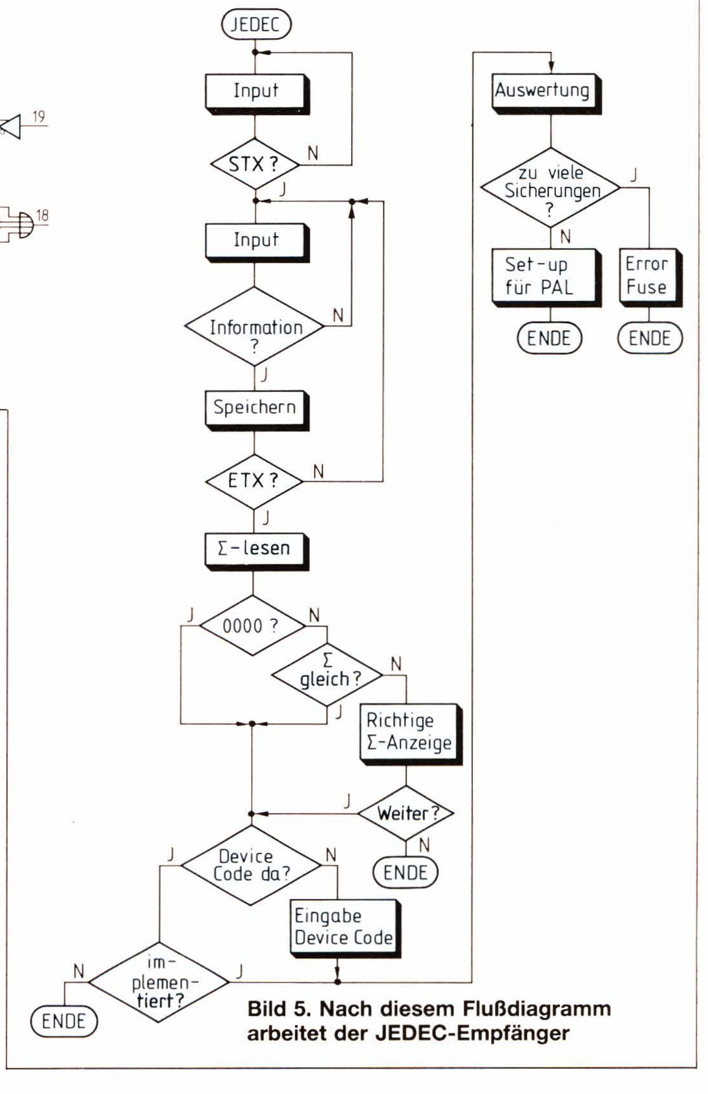


Bild 5. Nach diesem Flußdiagramm arbeitet der JEDEC-Empfänger

rungszustände. Wo sich diese Sicherung im Programmier-Array befindet, muß vom PAL-Hersteller festgelegt und vom JEDEC-Empfänger im Programmiergerät entsprechend abgelegt werden, damit auch tatsächlich diese Sicherung programmiert wird. Da nur programmierbare Sicherungen übertragen werden, ist dies bei der Zählweise zu berücksichtigen. *Bild 4* zeigt einen Teil eines leeren PAL12H6.

Bei genauerem Hinsehen erkennt man fehlende Eingänge (10,11,14,15,18,19,22 und 23) und fehlende Produkt-Termen (0...7,12...15 u.s.w.). An den nicht vorhandenen Kreuzungspunkten sind zwar ebenfalls Sicherungen vorhanden, sie sind aber nicht programmierbar und können auch noch unterschiedliche Zustände haben.

Für die Berechnung einer Sicherungsnummer in einem PAL beginnt man bei "Eingang 0" und "Produkt-Term 0". Diese Sicherung ist aber beim 12H6 nicht programmierbar. Die nächste Sicherung befindet sich am Schnittpunkt "Eingang 1" und "Produkt-Term 0"; sie ist ebenfalls nicht programmierbar, da kein gezeichneter Kreuzungspunkt vorhanden ist (sog. „Phantom Fuses").

Schließlich findet man die erste programmierbare Sicherung am Schnittpunkt des Einganges 0 mit dem Produkt-Term 8. Diese Sicherung ist gemeint, wenn in einem JEDEC-Protokoll für das PAL12H6 für die Sicherung 0 ein Zustand übertragen wird. Setzt man den Zählvorgang fort, so gelangt man zur Sicherung 10 am Kreuzungspunkt von Term 8 mit Eingang 12 usw.

Vollzieht man denselben Vorgang für ein anderes PAL, so stellt man fest, daß hier die erste programmierbare Sicherung an einem anderen Schnittpunkt liegen kann. So liegt beim PAL14H4 die Sicherung 0 am Schnittpunkt des Einganges 0 mit dem Produkt-Term 16, beim PAL 16L8 ist dagegen die Sicherung 0 identisch mit der ersten Sicherung im Array, da alle Sicherungen programmierbar sind.

Der JEDEC-Empfänger

Der Aufbau eines JEDEC-Empfängers in einem Programmiergerät muß aufgrund des bisher Gesagten eigene "Intelligenz" besitzen und auch sehr fehlertolerant aufgebaut sein. Verschiedene Varianten des JEDEC-Protokolls verzichten z. B. auf das Device-Feld. Dadurch kann sich das Programmiergerät nicht automatisch auf den Typ einstellen.

Der Empfänger sollte es dem Bediener auch trotz einer fehlerhaften Übertragung (Fehler beim Vergleich der Prüfsumme) gestatten, die übertragenen Daten in Programmierdaten umzuwandeln. Bei einer fehlerhaften Übertragung, die beispielsweise nur durch eine falsch berechnete Prüfsumme entstanden ist, sollte durch manuellen Eingriff mit einer entsprechenden Fehlermeldung die Möglichkeit bestehen, mit der Übertragung fortzufahren. In *Bild 5* ist ein Flußdiagramm für den JEDEC-Empfänger dargestellt, wie er im "Universal Programming Module (UPM)" von Kontron realisiert ist.

Der Empfänger beginnt mit der Aufzeichnung aller ankommenden Zeichen an der seriellen Schnittstelle, sobald er ein "STX" erkennt. Alle für die spätere Auswertung benötigten Daten werden in einem Speicher abgelegt. Nach dem Erkennen von "ETX" liest der Empfänger noch vier Zeichen, die die Prüfsumme enthalten und vergleicht sie mit der intern errechneten. War die gesendete Summe 0000, so überspringt er diesen Ver-

gleich. Bei Ungleichheit wird die zu erwartende Summe angezeigt, dem Bediener aber die Möglichkeit gegeben, durch entsprechende Eingabe fortzufahren oder abzubrechen.

In den abgelegten Daten wird nun nach einem Device-Feld gesucht und mit der Auswertung fortgefahren, wenn der UPM den Device-Code unterstützt. Liegt ein ungültiger Typ vor, so erfolgt ein Abbruch mit einer entsprechenden Fehlermeldung. Bei einem nicht übertragenen Device-Code kann der Anwender einen PAL-Typ manuell eingeben. Entsprechend dem Device-Code setzt der Empfänger alle empfangenen Sicherungsinformationen in ein Programmier-Array um. Wurden aus irgendwelchen Gründen zuviele Sicherungen übertragen, erfolgt auch hier eine Fehlermeldung mit einem Abbruch.

Schließlich stellt sich das UPM auf den neuen Typ ein und überträgt das Programmier-Array in den Datenspeicher, aus dem dann das PAL programmiert werden kann.

Der Datenspeicher

Die momentane Größe des Datenspeichers ist abhängig von der Größe des Programmier-Arrays des momentan aktivierten PALs. Bei PALs im 20-Pin-Gehäuse läßt sich der Datenspeicher nur im Adreßbereich 000 bis 1FF benutzen; jeder andere Speicherzugriff ist durch das Betriebssystem im UPM-Modul unterbunden.

Das manuelle Verändern von Speicherinhalten ist möglich, jedoch nur dann, wenn programmierbare Sicherungen verändert werden. Dagegen kann eine "Phantom Fuse" nicht modifiziert werden, da diese nicht programmierbar ist. Deshalb sind auch andere Funktionen wie z. B. "Datenspeicher invertieren" oder "Füllen mit gleichen Daten" nicht eingebaut.

Testvektoren

Um mit einem Programmiergerät ein programmiertes PAL auf seine logische Funktion zu testen, benötigt man Testvektoren, die in einer bestimmten Reihenfolge an das PAL angelegt und gleichzeitig gelesen werden. Durch den Aufbau des JEDEC-Datenprotokolls ist es relativ einfach, Testvektoren an das Programmiergerät zu übertragen. Das geschieht ähnlich wie bei der Übertragung von Sicherungsinformation und könnte z. B. wie folgt aussehen:

V0001 XXXXXXXXXNXXXXXXXXH0N*
V0002 XXXXXXXXXNXXXXXXXXL1N*

Sinngemäß wird für die Kennzeichnung eines Vektorfeldes ein V und für die fortlaufende Nummer eine 4stellige Zahl, ähnlich der Sicherungsinformation, verwendet. Die folgenden Zeichen stehen symbolisch für eine Testbedingung an dem entsprechenden Anschluß. Nachstehend sind alle erlaubten Zeichen und ihre Bedeutung aufgeführt.

0 --- Eingang auf log.low bringen
1 --- Eingang auf log.high bringen
2-9 --- Eingang auf >ttl bringen (z. B. Register preload)
C --- Eingang mit Takt beschalten (low,high,low)
F --- Ein(Ausgang) in Tristate schalten
H --- Ausgang auf log.high testen
K --- Eingang mit Takt beschalten (high,low,high)
L --- Ausgang auf log.low testen
N --- Versorgungs- und nicht getestete Pins
P --- Setze Register
X --- Nicht zu testender Ausgang,Eingang ist offen
Z --- Eingang oder Ausgang auf tristate testen

Die beiden oben angeführten Testvektoren erlauben es, den im Beispiel enthaltenen Inverter zu testen.

Der Vektor 1 legt an Pin 19 eine 0 an und erwartet an Pin 18 ein H (das 18. Zeichen ist ein H, d. h. Pin 18 muß auf logisch 1 getestet werden, das 19. Zeichen ist ein 0, d. h. der Pin 19 muß auf logisch 0 geschaltet werden). Das gleiche gilt für den Vektor 2, nur mit umgekehrter Polarität. Diese Testvektoren können entweder vom PAL-Designer oder durch ein Simulationsprogramm erstellt werden.

Die "Last Fuse Option"

Durch ein zusätzliches Feld kann dem Programmiergerät mitgeteilt werden, ob die sogenannte "letzte Sicherung", die ein unerlaubtes Kopieren von PALs verhindert, programmiert werden soll oder nicht. Diese zusätzliche Sicherung unterbindet den Zugriff auf das Programmier-Array. Dadurch lassen sich keine funktionsgleichen Kopien mehr herstellen, die logische Funktion bleibt jedoch erhalten.

Mit der Zeichenfolge
G0*
wird dem Programmiergerät mitgeteilt, daß die letzte Sicherung nicht programmiert werden soll, dagegen programmiert "G1*" nach erfolgreicher Array-Programmierung und anschließender Verifizierung die letzte Sicherung.

Zwei zusätzliche Felder enthalten Informationen, wieviele Anschlüsse und Sicherungen das zu programmierende PAL hat:
QF280*
QP20*
Dies bedeutet, daß das PAL 280 programmierbare Sicherungen und 20 Anschlüsse hat. Ein "Intelligenter Empfänger" benötigt diese Information jedoch nicht, es läßt sich daraus auch nicht eindeutig der richtige Typ einstellen.

Konvertierung des Programmier-Arrays

Da unterschiedliche PAL-Hersteller auch unterschiedliche Programmier-Arrays benutzen, ist beim Umschalten zwischen funktionsgleichen PALs unterschiedlicher Hersteller das Array so umzukonvertieren, daß sich ohne zusätzliche Handgriffe ein funktionsgleiches PAL programmieren läßt.

EPROM als Datenspeicher für PAL-Daten

Beim Programmieren von PALs in der Produktion ist es notwendig, ein möglichst einfaches Datenmedium zu verwenden, um mit relativ geringem Aufwand das Programmier-Array für das entsprechende PAL zu generieren. Auf sogenannte Master-PALs mit einer Prüfsumme als Kennzeichnung sollte man hier lieber verzichten. Dieses Verfahren ist bei bipolaren PROMs und EPROMs populär, kann aber bei PALs zu Problemen führen. Verschiedene PAL-Hersteller verwenden nämlich unterschiedliche Algorithmen zum Auslesen des Programmier-Arrays, und der Fehler wird meist viel zu spät erkannt. Durch „intelligente" Ladealgorithmen kann man feststellen, ob irgend eine Phantom-Fuse nicht mit den zu erwartenden übereinstimmt. Verwendet man aber voll programmierbare PALs (z.B. 16L8), so steht diese Testmöglichkeit nicht zur Verfügung. Außerdem kann ein Master-PAL keine Testvektoren abspeichern.

Eine komfortable Möglichkeit für die Produktion bietet dagegen das Programmiermodul UPM. Von jedem momentanen, im Datenspeicher abgelegten Programmier-Array läßt sich über einen EPROM-Sockel ein "Backup-EPROM" herstellen. Dadurch ergibt sich der Vorteil, in der Produktion nur EPROMs als PAL-Master archivieren zu müssen und mit wenigen Handgriffen ein PAL programmieren zu können. Gleichzeitig bietet das relativ billige Backup-Medium noch die Möglichkeit, ohne zusätzliche Kosten Testvektoren, einen 6stelligen Namen und den automatischen "Setup" für das entsprechende PAL abzuspeichern.

Nachfolgend der Inhalt eines EPROM's für das in diesem Beitrag verwendeten Beispiel BASIC GATES:

```
0000 E7 E0 E0 E0 FF FF E1 E2 F1 E6 FF FF FF 00 B0 DB
0010 43 21 FF 22 02 00 00 00 00 00 02 01 00 00 00 00
0020 0B 0F 0B 0F 07 0F 0F 0D 0F 0F 0F 0F 0F 0F 0F 0F
0030 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F
0040 09 09 09 09 09 09 09 09 01 09 0F 0F 09 09 0F 0F
0050 09 09 0F 0F 09 09 0F 0F 09 09 09 09 09 09 09 09
0060 01 01 01 01 01 01 01 01 01 01 03 03 01 01 03 03
0070 01 01 03 03 01 01 03 03 01 01 01 01 01 01 01 01
0080 01 01 01 01 01 01 01 01 01 01 03 03 01 01 03 03

0190 08 08 0C 0C 08 08 0C 0C 08 08 08 08 08 08 08 08
01A0 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
01B0 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
01C0 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
01D0 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
01E0 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
01F0 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
0200 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
0210 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
0220 59 1D FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Die ersten sechs Bytes enthalten ein Identifikationswort, das der Anwender vor dem Backup eingeben kann. Hiermit meldet sich das Programmiergerät immer dann, wenn aus dem EPROM ein PAL-Array generiert werden soll. Es handelt sich also quasi um ein programmiertes Etikett, das die Archivierung erleichtert. Die folgenden sieben Bytes kennzeichnen den PAL-Typ. Im nächsten Byte versteckt sich der Herstellercode für das PAL. Danach kommen zwei Bytes mit der Prüfsumme, deren Berechnungsalgorithmus im darauffolgenden Byte codiert ist. Das nächste Byte enthält den Typ des verwendeten EPROMs. Die folgenden zwei Bytes enthalten die Länge des gesamten Datenfeldes, das im EPROM abgespeichert ist. Vier weitere Bytes sind für die Verwaltung von Testvektoren vorgesehen. Zwei Bytes enthalten die Größe des verwendeten Programmier-Arrays für das PAL. Ein zusätzliches Byte wird noch benötigt, um die interne Umkonvertierung zu vereinfachen. Die restlichen vier Bytes sind für zusätzliche Informatinen reserviert.

Die folgenden Speicherplätze sind eine exakte Kopie aus dem Datenspeicher; ihre Anzahl entspricht dem Umfang des Programmier-Arrays. Zur Datensicherheit folgt dann noch eine Prüfsumme. Sie sorgt dafür, daß nur gültige Daten aus dem EPROM in den Datenspeicher für das Array übertragen werden. Stimmt etwas nicht, so erfolgt eine Fehlermeldung und das momentane Programmier-Array bleibt erhalten.

Der "Fuse Plot"

Eine sehr nützliche Zusatzfunktion bei der PAL-Programmierung kann mitunter die Erzeugung eines sogenannten "Fuse Plots" sein. Damit kann man das Innenleben eines PALs in gut lesbarer Form zu Papier bringen. Der folgende – aus Platzgründen verkürzte – Ausdruck entspricht wieder dem Beispiel BASIC GATES.

```
PAL MMI 12H6

               11 1111 1111 2222 2222 2233
0123 4567 8901 2345 6789 0123 4567 8901
 8 ---- ---X --00 --00 --00 --00 ---- ----
 9 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
10 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
11 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

16 X-X- ---- --00 --00 --00 --00 ---- ----
17 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

24 ---- X--- --00 --00 --00 --00 ---- ----
25 ---- ---- X-00 --00 --00 --00 ---- ----

32 ---- ---- --00 -X00 -X00 --00 ---- ----
33 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

40 ---- ---- --00 --00 --00 X-00 -X-- ----
41 ---- ---- --00 --00 --00 -X00 X--- ----

48 ---- ---- --00 --00 --00 --00 ---- -X--
49 ---- ---- --00 --00 --00 --00 ---- ---X
50 ---- ---- --00 --00 --00 --00 ---X ----
51 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

Legend:        fuse blown       : -
               fuse intact      : X
               phantom fuse  0  : o
               phantom fuse  1  : 0

Fuses blown: 226
```

Phantom-Fuse:

Hierunter versteht man Sicherungen, die bei MMI-PALs an den Kreuzungspunkten liegen, die nur zum Auffüllen des Programmierarrays vorhanden sein müssen. Damit ergibt sich für alle PALs mit gleicher Pin-Zahl immer die gleiche Arraygröße.

Willibald Voldan

PAL als „Memory Decoder"

Speicher gezielt ansprechen

Die einfachste und auch oft verwendete Applikation für PAL-Bausteine ist die eines Decoders, der z. B. die „Chip-Select"-Signale für verschiedene Speicherbereiche eines Rechnersystems erzeugt.

Nach der Programmierung des Bausteins mit einem dementsprechenden Pattern decodiert das programmierbare Array des PALs die Speicher-Adressen und selektiert Eingabe-/Ausgabe-Kanäle aus dem Adreßbereich. Sämtliche Adressen und Steuer-Signale, die zur Bildung der Select-Signale notwendig sind, werden dabei als Eingänge in den PAL-Baustein geführt.

PAL-Eingänge und auch etwaige Rückkoppel-Eingänge werden immer invertiert und nicht invertiert in die programmierbare Matrix des Bausteins geführt. Durch diese Eigenschaft der PALs kann jede beliebige Adresse oder ein Adreß-Bereich decodiert werden.

Durch einfaches Editieren der jeweiligen Gleichungen lassen sich die verschiedenen Systemkomponenten im Adreß-Bereich der Schaltung verschieben. Somit ist größtmögliche Flexibilität und Anpassungsmöglichkeit unter Verwendung der PAL-Bausteine gewährleistet.

```
TITLE    Memory_Decoder
PATTERN  MemIO.pds
REVISION A
AUTHOR   A.G.Gilbert
COMPANY  Monolithic Memories Inc., Santa Clara, CA
DATE     01/08/1985

;This Pal Design Specification demonstrates a typical design for
;a memory decoder (chip select).
;Personal computers which are hardware compatible with the
;ubiquitous IBM PC share this I/O map.

CHIP PC_IO PAL8L14

NC NC A9 A8 A7 A6 A5 A4 A3 AEN /CSMONO GND
/CSGAME /CSCOLOR /CSPRINT /CSFLOPPY /CSRS232 /CSNMI
/CSDMA  /CSPCHIP /CSTIMER /CSINCHIP /CSDCHIP  VCC

EQUATIONS

CSDCHIP  = /A9*/A8*/A7*/A6*/A5*/A4*     /AEN  ;DMA Controller
                                              ;Hex cddress 000-00F
CSINCHIP = /A9*/A8*/A7*/A6* A5*/A4*/A3*/AEN   ;INTERRUPT Controller
                                              ;Hex address 020-021
CSTIMER  = /A9*/A8*/A7* A6*/A5*/A4*/A3*/AEN   ;TIMER
                                              ;Hex address 040-043
CSPCHIP  = /A9*/A8*/A7* A6* A5*/A4*/A3*/AEN   ;PARALLEL PRINTER INT.
                                              ;Hex address 060-063
CSDMA    = /A9*/A8* A7*/A6*/A5*/A4*/A3*/AEN   ;DMA PAGE REGISTER
                                              ;Hex address 080-083
CSNMI    = /A9*/A8* A7*/A6* A5*/A4*/A3*/AEN   ;NMI MASK REGISTER
                                              ;Hex address 0AX
CSRS232  =  A9* A8* A7* A6* A5* A4* A3*/AEN   ;RS232 MODULE
                                              ;Hex address 3F8-3FF
CSFLOPPY =  A9* A8* A7* A6* A5* A4*/A3*/AEN   ;FLOPPY DISK MODULE
                                              ;Hex address 3F0-3F7
CSPRINT  =  A9*/A8*/A7* A6* A5* A4* A3*/AEN   ;PARALLEL PRINTER MOD.
                                              ;Hex address 378-37F
CSCOLOR  =  A9* A8* A7* A6*/A5* A4*     /AEN  ;COLOR GRAPHICS MOD.
                                              ;Hex address 3D0-3DF
CSGAME   =  A9*/A8*/A7*/A6*/A5*/A4*     /AEN  ;GAME I/O MODULE
                                              ;Hex address 200-20F
CSMONO   =  A9* A8* A7*/A6* A5* A4*     /AEN  ;MONOCHROME VIDEO MOD.
                                              ;Hex address 3B0-3BF
```

Bild 1. Das Programm eines typischen Speicher-Decoders

```
SIMULATION

TRACE_ON A9 A8 A7 A6 A5 A4 A3 AEN
         /CSMONO /CSGAME /CSCOLOR /CSPRINT /CSFLOPPY /CSRS232
         /CSNMI  /CSDMA  /CSPCHIP /CSTIMER /CSICHIP  /CSDCHIP

SETF AEN                                   ;set outputs to HIGH
SETF /A9 /A8 /A7 /A6 /A5 /A4 /A3 /AEN      ;select CSDCHIP
SETF                     A5                ;select CSICHIP
SETF                 A6                    ;select CSPCHIP
SETF                    /A5                ;select CSTIMER
SETF             A7 /A6                    ;select CSDMA
SETF                     A5                ;select CSNMI
SETF  A9  A8         A6      A4  A3        ;select CSRS232
SETF                            /A3        ;select CSFLOPPY
SETF                    /A5                ;select CSCOLOR
SETF                /A6  A5                ;select CSMONO
SETF     /A8 /A7        /A5 /A4            ;select CSGAME
SETF                 A6  A5  A4  A3        ;select CSPRINT

;Simulation part of former PALASM-I with function table vectors
;
;FUNCTION TABLE
;
;A9 A8 A7 A6 A5 A4 A3 AEN
;/CSMONO /CSGAME /CSCOLOR /CSPRINT /CSFLOPPY /CSRS232
;/CSNMI  /CSDMA  /CSPCHIP /CSTIMER /CSICHIP  /CSDCHIP
;-----------------------------------------------------------------
; X X X X X X X  H   H H H H H H H H H H H H ;set outputs to HIGH
; L L L L L L L  L   H H H H H H H H H H H L ;select CSDCHIP
; L L L L H L L  L   H H H H H H H H H H L H ;select CSICHIP
; L L L H H L L  L   H H H H H H H H L H H H ;select CSPCHIP
; L L L H L L L  L   H H H H H H H H L H H H ;select CSTIMER
; L L H L L L L  L   H H H H H H H L H H H H ;select CSDMA
; L L H L H L L  L   H H H H H H L H H H H H ;select CSNMI
; H H H H H H H  L   H H H H H L H H H H H H ;select CSRS232
; H H H H H H L  L   H H H H L H H H H H H H ;select CSFLOPPY
; H H H H L H L  L   H H L H H H H H H H H H ;select CSCOLOR
; H H H L H H L  L   L H H H H H H H H H H H ;select CSMONO
; H L L L L L L  L   H L H H H H H H H H H H ;select CSGAME
; H L L H H H H  L   H H H L H H H H H H H H ;select CSPRINT
------------------------------------------------------------------
```

Bild 2. Und hiermit läßt sich der Decoder in seiner Funktion simulieren

Willibald Voldan

Schneller DMA-Controller mit PALs

Diese Applikation beschäftigt sich mit der Definition eines allgemeinen DMA-Controllers, der aus fünf PAL-Bausteinen aufgebaut ist. Der Controller kann Blöcke von Daten mit besonders hoher Geschwindigkeit zwischen der Peripherie und dem Speicher transportieren.

Daten werden von Peripherie-Bauelementen in Blöcken zu max. 64 KBit ab einer vorgegebenen Startadresse aus dem Speicher gelesen oder in den Speicher geschrieben. Die Startadresse, die Blocklänge und die Instruktions-Anweisung werden vor jedem Datentransfer in das PAL geschrieben.

Nach einer DMA-Anfrage von der Peripherie an den Controller gibt dieser das Signal an die CPU weiter, um die Kontrolle über den Bus übernehmen zu können. Wird der Daten-, Adreß- und Steuer-Bus von der CPU freigegeben und in den Three-State-Zustand geschaltet,

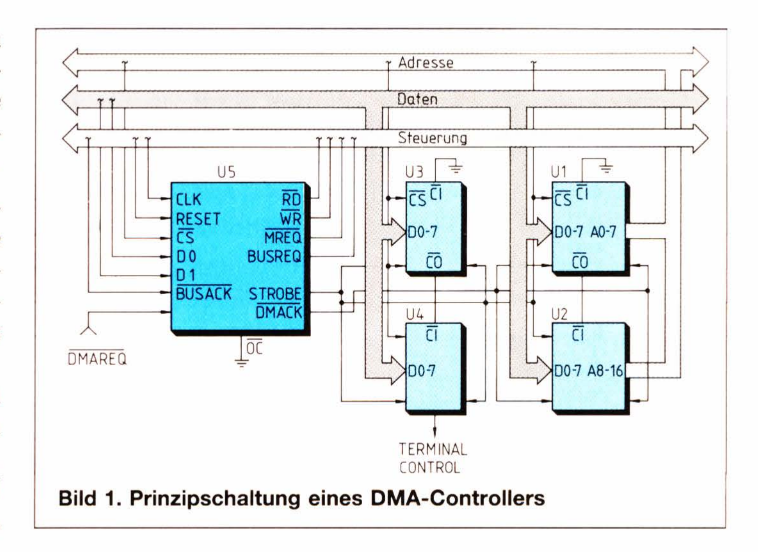


Bild 1. Prinzipschaltung eines DMA-Controllers

```
PAL20X8                                          PAL DESIGN SPECIFICATIONS
DMA2.DAT                                         DANESH TAVANA
DMA LENGTH COUNTER
MMI SUNNYVALE
   CLK /CI D0 D1 D2 D3 D4 D5 D6 D7 STROBE GND
/DMACK /CS L7 L6 L5 L4 L3 L2 L1 L0    /CO VCC

IF(VCC)  CO  = /L7*/L6*/L5*/L4*/L3*/L2*/L1*/L0          ;CARRY OUT TERM

      /L0 :=  CS*/D0                                    ;LOAD
           + /CS*/L0                                    ;HOLD
          :+:/CS* STROBE* CI                            ;INCREMENT

      /L1 :=  CS*/D1                                    ;LOAD
           + /CS*/L1                                    ;HOLD
          :+:/CS* STROBE* CI*/L0                        ;INCREMENT

      /L2 :=  CS*/D2                                    ;LOAD
           + /CS*/L2                                    ;HOLD
          :+:/CS* STROBE* CI*/L0*/L1                    ;INCREMENT

      /L3 :=  CS*/D3                                    ;LOAD
           + /CS*/L3                                    ;HOLD
          :+:/CS* STROBE* CI*/L0*/L1*/L2                ;INCREMENT

      /L4 :=  CS*/D4                                    ;LOAD
           + /CS*/L4                                    ;HOLD
          :+:/CS* STROBE* CI*/L0*/L1*/L2*/L3            ;INCREMENT

      /L5 :=  CS*/D5                                    ;LOAD
           + /CS*/L5                                    ;HOLD
          :+:/CS* STROBE* CI*/L0*/L1*/L2*/L3*/L4        ;INCREMENT

      /L6 :=  CS*/D6                                    ;LOAD
           + /CS*/L6                                    ;HOLD
          :+:/CS* STROBE* CI*/L0*/L1*/L2*/L3*/L4*/L5    ;INCREMENT

      /L7 :=  CS*/D7                                    ;LOAD
           + /CS*/L7                                    ;HOLD
          :+:/CS* STROBE* CI*/L0*/L1*/L2*/L3*/L4*/L5*/L6 ;INCREMENT
```

Bild 2. Dieses PALASM-Programm macht aus einem PAL20X8 einen Zähler für die Blocklänge

```
PAL20X8                                          PAL DESIGN SPECIFICATIONS
DMA1.DAT                                         DANESH TAVANA
DMA ADDRESS COUNTER
MMI SUNNYVALE
   CLK /CI D0 D1 D2 D3 D4 D5 D6 D7 STROBE GND
/DMACK /CS A7 A6 A5 A4 A3 A2 A1 A0    /CO VCC

IF(VCC)  CO  =  A7* A6* A5* A4* A3 *A2 *A1 *A0         ;CARRY OUT TERM

      /A0 :=  CS*/D0                                   ;LOAD
           + /CS*/A0                                   ;HOLD
          :+:/CS* STROBE* CI                           ;INCREMENT

      /A1 :=  CS*/D1                                   ;LOAD
           + /CS*/A1                                   ;HOLD
          :+:/CS* STROBE* CI*A0                        ;INCREMENT

      /A2 :=  CS*/D2                                   ;LOAD
           + /CS*/A2                                   ;HOLD
          :+:/CS* STROBE* CI*A0*A1                     ;INCREMENT

      /A3 :=  CS*/D3                                   ;LOAD
           + /CS*/A3                                   ;HOLD
          :+:/CS* STROBE* CI*A0*A1*A2                  ;INCREMENT

      /A4 :=  CS*/D4                                   ;LOAD
           + /CS*/A4                                   ;HOLD
          :+:/CS* STROBE* CI*A0*A1*A2*A3               ;INCREMENT

      /A5 :=  CS*/D5                                   ;LOAD
           + /CS*/A5                                   ;HOLD
          :+:/CS* STROBE* CI*A0*A1*A2*A3*A4            ;INCREMENT

      /A6 :=  CS*/D6                                   ;LOAD
           + /CS*/A6                                   ;HOLD
          :+:/CS* STROBE* CI*A0*A1*A2*A3*A4*A5         ;INCREMENT

      /A7 :=  CS*/D7                                   ;LOAD
           + /CS*/A7                                   ;HOLD
          :+:/CS* STROBE* CI*A0*A1*A2*A3*A4*A5*A6 ;INCREMENT
```

Bild 3. Ebenfalls ein PAL20X8 dient als Zähler für den Adreßbereich

```
PAL20X10                                             PAL DESIGN SPECIFICATIONS
DMA3.DAT                                             DANESH TAVANA
DMA CONTROLLER PAL
MMI SUNYVALE
CLK /RESET     /CS     D0 D1 NC NC NC  NC /DMAREQ /BUSACK GND
/OC /BUSREQ /DMACK STROBE T1 T0 I1 I0 /WR     /RD   /MREQ VCC



  MREQ := /RESET* MREQ                       ;HOLD
        + /RESET* MREQ                       ;HOLD
       :+:/RESET* DMACK*/T1*/T0              ;SET ON FIRST CLOCK IF DMACK
        + /RESET* DMACK* T1* T0              ;RESET ON FOURTH CLOCK IF DMACK

    RD := /RESET* RD                         ;HOLD
        + /RESET* RD                         ;HOLD
       :+:/RESET* MREQ*/T1* T0* I1* I0       ;SET ON SECOND CLOCK IF READ INST.
        + /RESET* MREQ* T1* T0* I1* I0       ;RESET ON FOURTH CLOCK

    WR := /RESET* WR                         ;HOLD
        + /RESET* WR                         ;HOLD
       :+:/RESET* MREQ* T1*/T0*/I1*/I0       ;SET ON THIRD CLOCK IF WRITE INST.
        + /RESET* MREQ* T1* T0*/I1*/I0       ;RESET ON THIRD CLOCK

   /I0 :=  CS*/D0                            ;LOAD DATA IF CHIP SELECTED
        +  CS*/D0                            ;LOAD DATA IF CHIP SELECTED
       :+:/CS*/I0                            ;HOLD INSTRUCTION OTHERWISE

   /I1 :=  CS*/D1                            ;LOAD DATA IF CHIP SELECTED
        +  CS*/D1                            ;LOAD DATA IF CHIP SELECTED
       :+:/CS*/I1                            ;HOLD INSTRUCTION OTHERWISE

   /T0 := /RESET*/T0                         ;STATE COUNTER
        +  RESET                             ;RESET
       :+:/RESET* DMACK                      ;START COUNTER IF DMACK

   /T1 := /RESET*/T1                         ;STATE COUNTER
        +  RESET                             ;RESET
       :+:/RESET* DMACK* T0                  ;START COUNTER IF DMACK

/STROBE := /RESET*/STROBE                    ;HOLD
        + /RESET*/STROBE                     ;HOLD
       :+:/RESET* DMACK*/T1*/T0              ;SET ON FIRST CLOCK IF DMACK
        + /RESET* DMACK* T1* T0              ;RESET ON FOURTH CLOCK

 DMACK := /RESET* DMACK                      ;HOLD
        + /RESET* DMACK                      ;HOLD
       :+:/RESET*/DMACK * BUSACK             ;SET IF BUS IS ACKNOWLEDGED
        + /RESET*/DMAREQ* BUSREQ             ;RESET IF NO DMA REQUEST

BUSREQ := /RESET* BUSREQ                     ;HOLD
        + /RESET* BUSREQ                     ;HOLD
       :+:/RESET* DMAREQ*/BUSREQ             ;SET IF THERE IS DMA REQUEST
        + /RESET*/DMAREQ* BUSREQ             ;RESET IF NO DMA REQUEST
```

Bild 4. Die Steuerung des DMA-Controllers übernimmt PAL20X10

übernimmt der PAL-Controller die Steuerung und führt die vorgegebene Instruktion aus.

Nach Beendigung des Datentransfers gibt ein Signal die Kontrolle an die CPU zurück.

Vier PALs dienen als Zähler für die Adresse und für die Blocklänge. Die Zählerstufen sind programmierbar und kaskadierbar. Sie können vorwärts und rückwärts zählen. Das fünfte PAL erzeugt die notwendigen Steuersignale, um den Controller an die jeweilige CPU-Architektur anzupassen.

Nachfolgend sind die Aufgaben des DMA-Controllers in der Reihenfolge nach einem „DMA-REQUEST“ zusammengefaßt:

- Die CPU lädt das niederwertige Byte der Startadresse in PAL#1.
- Die CPU lädt das höherwertige Byte der Startadresse in PAL#2.
- Die CPU lädt das niederwertige Byte der Blocklänge in PAL#3.
- Die CPU lädt das höherwertige Byte der Blocklänge in PAL#4.
- Die CPU lädt den Instruktions-Code in PAL#5.
- Der Bus wird im Normalzustand von der CPU gesteuert und das Interface des PAL-DMA-Controllers ist im Three-State-Zustand. Der DMACK-Ausgang des PALs wird zur Freigabe der DMA-Adresse und der Steuersignale verwendet, wenn ein BUSACK-Signal von der CPU empfangen wird.
- Der Block-Transfer zum Schreiben oder Lesen der Daten von einer bestimmten Startadresse ab wird ausgeführt. Das „CARRY-OUT“-Signal des PALs zeigt das Ende der Übertragung an.
- Die Peripherie reagiert auf dieses Signal mit der Freigabe der DMA-Anfrage und gibt die Kontrolle wieder an die CPU zurück.

Alfie Gilbert, Gerhard Kirschner

Der „Frame Grabber“ Digitale Bildverarbeitung mit PALs

Das folgende Anwendungsbeispiel beschreibt das Gegenstück zu einem Video-Controller, nämlich einen sogenannten „Frame Grabber“. Ein Video-Controller wandelt eine Computer-Standardschnittstelle (z. B. RS232) in ein Fernsehsignal um. Der „Frame Grabber“ dagegen bringt das Fernsehsignal über eine Standardschnittstelle in den Speicher des Rechners und somit auf den Bildschirm.

Die Applikation wurde auf der Grundlage der in den USA gültigen Fernsehnormen verfaßt. Da diese Anwendung speziell für IBM-kompatible Personal Computer ausgelegt wurde, sind die Werte des OSC-Signals mit 14,318 MHz für dieses Beispiel entscheidend. Dieses Signal liegt am Anschluß B 30 des internen Busses des IBM PC an. Die meisten der in Deutschland erhältlichen Videokameras, speziell die für industrielle Anwendungen, liefern ein Ausgangssignal nach RS-170A wie das in der Applikation verwendete. Um jedoch die Unterschiede zu dem in Deutschland gültigen Standard B,G der CCIR-Norm mit dem für die USA zutreffenden Standard M aufzuzeigen, soll folgende Tabelle dienen:

	CCIR-Norm	
	B,G (BRD)	M (USA)
Zeilenzahl	625	525
Rasterwechselfrequenz	50 Hz	60 Hz
Zeilenfrequenz	15 625 Hz	15 750 Hz
Videobandbreite	5 MHz	4,2 MHz
Zeilendauer	64 µs	63,5 µs
Halbbilddauer	20 ms	16,66 ms

Grundlagen der Bildübertragung

Als Ausgangspunkt der folgenden Darstellungen wird eine normale Videokamera (VHS- oder Beta-Format) auf ein Motiv wie das in *Bild 1* gezeigte gerichtet. Die Ausgangssignale solcher Videokameras wurden vor einigen Jahren von der EIA standardisiert und in einem Dokument mit der Bezeichnung RS-170A niedergelegt. Diese Video-Signale werden oft als „NTSC Composite Video“ bezeichnet. Grund dafür ist, daß sie vier Signal-Komponenten enthalten, die zusammengemischt werden. Diese vier Signale sind: Luminanz (Helligkeit), Chrominanz (Farbsättigung), Audio (Ton) sowie das Synchronisationssignal. Diese Signale reichen aus, um Monitore direkt anzusteuern; sie bilden die Grundlage für Fernsehübertragungen.

Doch untersuchen wir einmal das Video-Ausgangssignal, das von einer Kamera erzeugt wird, die auf das „Graue-Streifen“-Motiv gerichtet ist. Die in *Bild 2* dargestellte Impulsfolge ist der Signalstrom, der durch die Abtastung vom Punkt A zum Punkt B erzeugt wird. Ein wichtiger Punkt, auf den hier besonders hingewiesen werden soll, ist, daß das Kamera-Ausgangssignal proportional zur Helligkeitsverteilung der Bildvorlage während des Abtastvorganges von A nach B ist. Dieser Prozeß der Abtastung von links nach rechts läuft 262,5mal ab, während das Motiv von oben nach unten durchlaufen wird. Zwischen jeder Abtastung liegt ein Zeitintervall, die sogenannte horizontale Rückführung

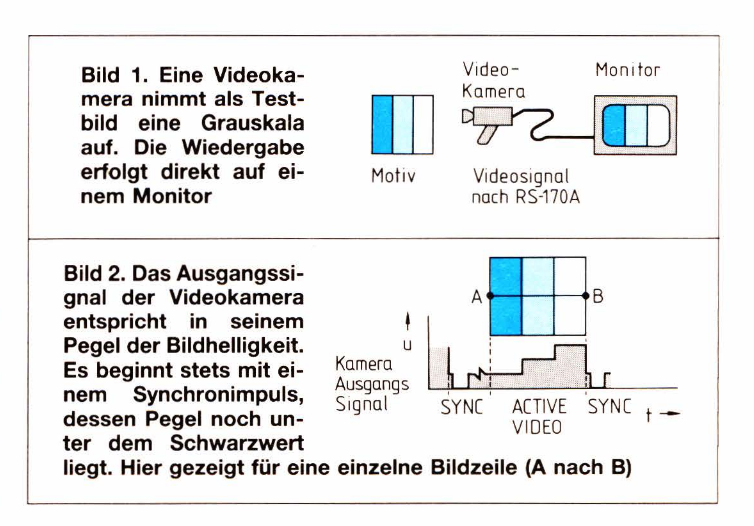


Bild 1. Eine Videokamera nimmt als Testbild eine Grauskala auf. Die Wiedergabe erfolgt direkt auf einem Monitor

Bild 2. Das Ausgangssignal der Videokamera entspricht in seinem Pegel der Bildhelligkeit. Es beginnt stets mit einem Synchronimpuls, dessen Pegel noch unter dem Schwarzwert liegt. Hier gezeigt für eine einzelne Bildzeile (A nach B)

(oder H sync). Während des Strahlrücklaufes wird der Abtaststrahl dunkel gesteuert, bis er ganz knapp unterhalb Punkt A wieder ansetzt. Der Abtastprozeß beginnt darauf von neuem.

Tabelle 1

```
TITLE      HOST BUS INTERFACE/CONTROL REGISTER
PATTERN    PALHB6.PDS
REVISION   A
AUTHOR     ALFIE GILBERT/WILLY VOLDAN
COMPANY    MONOLITHIC MEMORIES INC.
DATE       18.JULI 1985

CHIP INTERFACE PAL20RA10

 NC AEN /IOW    BA9   BA8  BA7    BA6  BA5 BA4  D0     D1 GND
/OE D2  /FBRRD /FBRWE INC /CLRADR MODE Q3 /IOR /245EN D3 VCC

; Diese PAL-Schaltung realisiert ein 4-Bit breites Steuerregister
; innerhalb der "FRAME-GRABBER" Anordnung. Die Ausgänge dieses
; Moduls reagieren auf die Adress-, Daten- und Steuerleitungen eines
; IBM-PC, bzw. eines dazu hardware kompatiblen Computers.
; Einige der Ausgänge sind kombinatorisch, andere aus offensicht-
; lichen Gründen mit Registern versehen. Dieses Modul erfüllt
; mehrere von einander unabhängige logische Funktionen und zeigt die
; enorme Flexibilität des asynchronen PALs 20RA10.

; MAKRO-DEFINITION für den Adreßbereich 110-11F HEX
STRING ADRESSE '/AEN*/BA9*BA8*/BA7*/BA6*/BA5'
;----------------------------------------
EQUATIONS

;  "FRAME-GRABBER" - Steuer Register Teil
;==============================================================

;  logische Gleichung für das Steuer-Register "INC"-Bit:
;  Ausgang uebernimmt das Signal D0 der Daten-Bus-Leitung am PC,
;  sobald ein I/O Schreibbefehl auf Adresse 110 bis 11F HEX erfolgt.
INC           := D0
INC.CLKF       = ADRESSE * IOW * BA4 ;siehe Makro-Definition
;-------------------------------------------------------

;  logische Gleichung für das Steuer-Register "/CLRADR"-Bit:
;  Ausgang uebernimmt das Signal D1 der Daten-Bus-Leitung am PC,
;  sobald ein I/O Schreibbefehl auf Adresse 110 bis 11F HEX erfolgt.
/CLRADR       :=/D1
/CLRADR.CLKF   = ADRESSE * IOW * BA4 ;siehe Makrodefinition
;------------------------------------------------------

;  logische Gleichung für das Steuer-Register "MODE"-Bit:
;  Ausgang uebernimmt das Signal D2 der Daten-Bus-Leitung am PC,
;  sobald ein I/O Schreibbefehl auf Adresse 110 bis 11F HEX erfolgt.
MODE          := D2
MODE.CLKF      = ADRESSE * IOW * BA4 ;siehe Makrodefinition
;------------------------------------------------------

;  logische Gleichung für das Steuer-Register-Bit
;  Reagiert auf einen I/O Schreibbefehl auf Adresse 11X HEX
Q3            := D3
Q3.CLKF        = ADRESSE * IOW * BA4 ;siehe Makrodefinition
;------------------------------------------------------
;  COMPUTER EIN/AUSGABE DECODER
;==============================================================

;  logische Gleichung für die Bus-Freigabe des 74LS245:
;  Dieser kombinatorische Ausgang des PAL20RA10 gibt bei
;  Adresse 100 bis 11F HEX den Bus-Transceiver 74LS245 frei.
245EN          = ADRESSE ;siehe Makrodefinition
;------------------------------------------

;  logische Gleichung für den Speicher-Schreibvorgang:
;  Dieser kombinatorische Ausgang fuellt die Speicherzelle des
;  RAM, auf die der Adress-Generator zeigt, wenn der Computer
;  auf die I/O Adresse 10X HEX schreibt.
FBRWE          = ADRESSE * IOW * /BA4
;---------------------------------

;  logische Gleichung für den Speicher-Lesevorgang:
;  Dieser kombinatorische Ausgang liest die Speicherzelle des
;  RAM, auf die der Adress-Generator zeigt, wenn der Computer
;  auf der I/O Adresse 10X HEX steht.
FBRRD          = ADRESSE * IOR * /BA4
;---------------------------------


SIMULATION

TRACE-ON  AEN /IOR /IOW BA9 BA8 BA7 BA6 BA5 BA4 D3 D2 D1 D0
          Q3 MODE /CLRADR INC /FBRWE /FBRRD /245EN

SETF OE /AEN /IOR IOW /BA9 BA8 /BA7 /BA6 /BA5 BA4 D3 D2 D1 D0
SETF /IOW /D0
SETF  IOW /D2                               ;CLOCK CONTROL REG
SETF /IOW /D3 /D1
SETF  IOW  D2  D0
SETF OE /AEN /IOR IOW /BA9 BA8 /BA7 /BA6 /BA5 BA4 /D3 /D2 /D1 /D0

SETF /IOR  IOW /BA4                         ;ASSERT /FBRWE

SETF  IOR /IOW /BA4                         ;ASSERT /FBRRD

SETF /IOR /IOW /BA9 BA8 /BA7 /BA6 BA5       ;ASSERT  /245EN
SETF  BA6 /BA5
SETF  BA7 /BA6                              ;UNASSERT
SETF /BA8 /BA7                              ;ACTIVE ADDRESS
SETF  BA9  BA8                              ;RANGE
SETF  AEN /BA9
```

Die Zeitperiode des Rücklaufes ist am Kamera-Ausgangssignal in Bild 2 als mit „Sync" bezeichneter Takt erkennbar. Eine Abtastung und Rückführung (Zeilendauer) geschieht in 63,5 µs. Eine Abfolge von 262,5 Abtastvorgängen nennt man ein Halbbild. Sie stellt einen vollständigen Durchlauf durch die Bildvorlage von oben nach unten dar. Hier nun setzt die Abtastung für einen verhältnismäßig langen Zeitraum aus, in dem der Strahl an den Anfang des Motivs zurückkehrt (Vertikalablenkung). Das gesamte Feld wird daraufhin erneut durchfahren, wobei jedoch die Abtastung des zweiten Feldes um eine halbe Zeile versetzt ist. Zwei aufeinanderfolgende Halbbilder ergeben das Gesamtbild. Und genau diese Informationsmenge wird von der nachfolgend beschriebenen Schaltung digitalisiert und im RAM abgespeichert. Sie stellt die Minimalinformation dar, die zur Rekonstruktion einer Abbildung benötigt wird.

Die Gesamtzahl der Horizontalabtastungen in einem Gesamtraster ist 262,5 Abtastungen/Halbbild mal 2 Halbraster ergibt 525 Zeilen. Dieses sogenannte Zeilensprungverfahren, also die Aufteilung eines vollständigen Abtastvorganges einer Bildvorlage in zwei versetzte Halbbilder, verringert das Flimmern des Bildes ganz wesentlich.

In Bildübertragungssystemen ist die richtige Zeitabstimmung ein kritischer Faktor. Video ist ein „hochrepetitiver" (sich oft wiederholender) Vorgang, und jedes Ereignis in diesem Prozeß geschieht synchron mit einem Takt. Der Takt für Composite Video ist normalerweise ein ganzzahliges Vielfaches (Oberschwingung) der NTSC-Farb-Burst-Frequenz. Die Farb-Burst-Frequenz ist von der EIA mit 3,579545 MHz festgelegt, und die Zeilenfrequenz des Composite-Video-Signals beträgt 15 750 Hz (Zeilendauer [H] = 63,556 µs).

Grundlagenberechnung des „Frame Grabbers"

An den internen Steckern der Buserweiterungen von Personal Computern, die mit dem IBM PC hardwarekompatibel sind, liegt ein Signal OSC mit einer Frequenz von 14,318 MHz an. Dieser Wert ist genau die 4. Harmonische des Farb-Bursts. Da diese Applikation für einen IBM PC verfaßt ist, verwendet man natürlich dessen OSC als den Systemtakt. Die Geschwindigkeit des A/D-Umsetzers entspricht bei diesem Beispiel der Frequenz des Farb-Bursts.

Ein paar kurze Berechnungen über das Zeitverhalten des „Frame Grabbers":

Bei einer Rasterwechselfrequenz von 30 Hz und einem Systemtakt von ca. 14,3 MHz vergehen 476 190 Systemtakt-Perioden in einem Rasterfeld. Da man einen 19-Bit-Zähler benötigt, um bis 476 190 zählen zu können, ist das Herzstück des „Frame Grabbers" in der Tat ein Zähler dieser Länge. Bei Digitalisierung des Video-Signals mit einer Genauigkeit von 2 Bit pro A/D-Umsetzung und einer Frequenz von 3,58 MHz werden 119 050 Abtastpunkte oder 238 100 Bit (entspricht etwa 32 KBit RAM) zur Abspeicherung des Informationsgehaltes

eines Bildrasters benötigt. Da der Systemtakt des „Frame Grabbers“ 14,3 MHz beträgt und die „Flash“-A/D-Umsetzfrequenz bei 3,58 MHz liegt, laufen zwischen den A/D-Umformungen vier Systemtakt-Zyklen (280 ns) ab.

Der Speicher ist 8 Bit breit organisiert, und der Umsetzer weist eine Unsicherheit von 2 Bit auf. Daher müssen vier aufeinanderfolgende Digitalisierungen in jedes Byte des RAMs „hineingepackt“ werden. Vier aufeinanderfolgende „Flashes“ brauchen für den vollständigen Ablauf 16 Systemtakt-Zyklen (1,12 μs). Ein 4-Bit-Zähler zählt stufenweise bis 16; so ist es natürlich, daß man den benötigten 19-Bit-Zähler aufteilt in einen 4-Bit-Zähler (der als Zeitbasis-Baustein in der MegaPAL-Schaltung ausgeführt wird) und einen 15-Bit-Zähler (Baustein zur Adreß-Generierung). Ein 15-Bit-Zähler zählt hoch bis 32 K, was dem benötigten Speicheradreß-Bereich zur Speicherung eines Bildrasters bei diesen speziellen Geschwindigkeiten und der gegebenen Auflösung entspricht.

Man kann diese Zahlen auch durch folgende Berechnung nachvollziehen: Ein Speicher-Schreibvorgang geschieht alle 1,12 μs (16 Zyklen des Systemtakts), und eine horizontale Abtastung (Zeilendauer) dauert 63,5 μs. Daher braucht man 58 Byte zur Speicherung einer Bildzeile. Ein Rasterbild hat 525 Zeilen, so daß der Adreßzähler während eines „freien“ Durchlaufs eines Rasterbildes auf 30 450 hochzählen wird.

Tabelle 2

```
TITLE       FLASH A/D CONVERTER
PATTERN     PALHB7.PDS
REVISION    A
AUTHOR      ALFIE GILBERT/WILLY VOLDAN
COMPANY     MONOLITHIC MEMORIES INC.
DATE        18. JULI 1985

CHIP        FLASH PAL16R8

 CLK /CLR Q358 Q716 NC T3 T2 T1 T0 GND
/OE   D0  D1   D2   D3 D4 D5 D6 D7 VCC

;
;         FLASH A/D MODUL
;
; Diese PAL-Schaltung ist ein "FLASH-A/D" Wandler mit geringer
; Auflösung. Vier Eingangs-Schwellwerte (T3-T0) werden synchron
; zu Gruppierungen von zwei Bit kodiert, die viermal "zusammen-
; gepackt" ein Ausgangsbyte bilden. Dieses Zusammenpacken wird
; von zwei parallelen 4-Bit Schieberegistern erledigt, die in
; diesem PAL realisiert sind.
; Ein Schieberegister arbeitet mit geradzahligen Bits, das andere
; mit ungeradzahligen (z.B. D0-D2-D4-D6).
; Diese CONVERSION/SHIFT Funktionen passieren bei der ansteigenden
; Flanke des 14.3818MHz System-Takts sobald die Eingangstakt-
; Signale der Zeitbasis den entsprechenden Zustand angenommen haben.
; Vier Taktperioden vergehen zwischen aufeinanderfolgenden Vor-
; gängen zur Abtastung und Umformung bei einer Taktrate von 3.58MHz.
; Wenn das geklammerte Video-Eingangssignal einen genügend starken
; Hub aufweist um die oberen drei Schwellwerte (weiß) zu über-
; schreiten, hat die 2-Bit Kodierung den Wert 11. Falls keiner der
; vier Schwellwerte (HSYNC) überschritten wird hat die Kodierung
; den Wert 00. Die erste Kodierung in einer Folge von vier Abtast-
; ungen wird im Register D7 und D6 abgespeichert, der letzte Kodier-
; schritt entsprechend in D1 und D0.
; Ein gleichmäßig ansteigendes Video-Signal von HSYNC nach Schwarz
; nach Grau nach Weiß wird vom Wert 00 01 10 11 dargestellt werden.
;
; Da das PAL16R8 keine programmierbare Ausgangspolarität hat
; ist das Design in negativer Logik ausgeführt.
;
EQUATIONS

/D0      :=/CLR * /T3 * /T2 * /T1 * /T0 * Q358 * Q716      ;CONVERT
          +/CLR *  T3 *  T2 * /T1 * /T0 * Q358 * Q716      ;CONVERT
          +/CLR * /Q358 * /D0                              ;HOLD
          +/CLR * /Q716 * /D0                              ;HOLD
          + CLR                                            ;CLEAR

/D1      :=/CLR * /T3 * /T2 * /T1 * /T0 * Q358 * Q716      ;CONVERT
          +/CLR *  T3 * /T2 * /T1 * /T0 * Q358 * Q716      ;CONVERT
          +/CLR * /Q358 * /D1                              ;HOLD
          +/CLR * /Q716 * /D1                              ;HOLD
          + CLR                                            ;CLEAR

/D2      :=/CLR *  Q358 * /D2                              ;HOLD
          +/CLR * /Q716 * /D2                              ;HOLD
          +/CLR * /Q358 *  Q716 * /D0                      ;SHIFT
          + CLR                                            ;CLEAR

/D3      :=/CLR *  Q358 * /D3                              ;HOLD
          +/CLR * /Q716 * /D3                              ;HOLD
          +/CLR * /Q358 *  Q716 * /D1                      ;SHIFT
          + CLR                                            ;CLEAR

/D4      :=/CLR *  Q358 * /D4                              ;HOLD
          +/CLR * /Q716 * /D4                              ;HOLD
          +/CLR * /Q358 *  Q716 * /D2                      ;SHIFT
          + CLR                                            ;CLEAR

/D5      :=/CLR *  Q358 * /D5                              ;HOLD
          +/CLR * /Q716 * /D5                              ;HOLD
          +/CLR * /Q358 *  Q716 * /D3                      ;SHIFT
          + CLR                                            ;CLEAR

/D6      :=/CLR *  Q358 * /D6                              ;HOLD
          +/CLR * /Q716 * /D6                              ;HOLD
          +/CLR * /Q358 *  Q716 * /D4                      ;SHIFT
          + CLR                                            ;CLEAR

/D7      :=/CLR *  Q358 * /D7                              ;HOLD
          +/CLR * /Q716 * /D7                              ;HOLD
          +/CLR * /Q358 *  Q716 * /D5                      ;SHIFT
          + CLR                                            ;CLEAR

;
;       PALASM1 FUNCTION TABLE DESCRIPTION
;
;                                8  6
; CONTROL      INPUTS            5  1            OUTPUTS
;/OE CLK /CLR T3 T2 T1 T0 Q3 Q7  D7 D6 D5 D4 D3 D2 D1 D0
;
;-----------------------------------------------------------
; L   C    L  X  X  X  X  X  X   L  L  L  L  L  L  L  L     CLEAR
; L   C    H  H  H  H  L  H  H   L  L  L  L  L  L  H  H     ENCODE WHITE
; L   C    H  X  X  X  X  L  L   L  L  L  L  L  L  H  H     HOLD
; L   C    H  X  X  X  X  L  H   L  L  L  L  H  H  H  H     SHIFT
; L   C    H  X  X  X  X  H  L   L  L  L  L  H  H  H  H     HOLD
; L   C    H  H  H  L  L  H  H   L  L  L  L  H  H  H  L     ENCODE GRAY
; L   C    H  X  X  X  X  L  L   L  L  L  L  H  H  H  L     HOLD
; L   C    H  X  X  X  X  L  H   L  L  H  H  H  L  H  L     SHIFT
; L   C    H  X  X  X  X  H  L   L  L  H  H  H  L  H  L     HOLD
; L   C    H  H  L  L  L  H  H   L  L  H  H  H  L  L  H     ENCODE BLACK
; L   C    H  X  X  X  X  L  L   L  L  H  H  H  L  L  H     HOLD
; L   C    H  X  X  X  X  L  H   H  H  H  L  L  H  L  H     SHIFT
; L   C    H  X  X  X  X  H  L   H  H  H  L  L  H  L  H     HOLD
; L   C    H  L  L  L  L  H  H   H  H  H  L  L  H  L  L     ENCODE HSYNC
; L   C    H  X  X  X  X  L  L   H  H  H  L  L  H  L  L     HOLD
; L   C    H  X  X  X  X  L  H   H  L  L  H  L  L  L  L     SHIFT
; L   C    H  X  X  X  X  H  L   H  L  L  H  L  L  L  L     HOLD
; L   X    X  X  X  X  X  X  X   Z  Z  Z  Z  Z  Z  Z  Z     TRISTATE
;-----------------------------------------------------------


SIMULATION
;
;
;
;
TRACE-ON T3 T2 T1 T0 Q358 Q716 D7 D6 D5 D4 D3 D2 D1 D0
SETF OE CLR                                     ;CLEAR
CLOCKF CLK
CHECK /D7 /D6 /D5 /D4 /D3 /D2 /D1 /D0

SETF /CLR T3 T2 T1 /T0 Q358 Q716                ;ENCODE WHITE LEVEL
CLOCKF CLK
SETF /Q358 /Q716                                ;HOLD
CLOCKF CLK
SETF /Q358 Q716                                 ;SHIFT
CLOCKF CLK
SETF Q358 /Q716                                 ;HOLD
CLOCKF CLK

SETF T3 T2 /T1 /T0 Q358 Q716                    ;ENCODE GRAY LEVEL
CLOCKF CLK
SETF /Q358 /Q716                                ;HOLD
CLOCKF CLK
SETF /Q358 Q716                                 ;SHIFT
CLOCKF CLK
SETF Q358 /Q716                                 ;HOLD
CLOCKF CLK

SETF T3 /T2 /T1 /T0 Q358 Q716                   ;ENCODE BLACK LEVEL
CLOCKF CLK
SETF /Q358 /Q716                                ;HOLD
CLOCKF CLK
SETF /Q358 Q716                                 ;SHIFT
CLOCKF CLK
SETF Q358 /Q716                                 ;HOLD
CLOCKF CLK

SETF /T3 /T2 /T1 /T0 Q358 Q716                  ;ENCODE HSYNC LEVEL
CLOCKF CLK
SETF /Q358 /Q716                                ;HOLD
CLOCKF CLK
SETF /Q358 Q716                                 ;SHIFT
CLOCKF CLK
SETF Q358 /Q716                                 ;HOLD
CLOCKF CLK

SETF /OE                                        ;TRISTATE
```

Die Ausgänge der Zeitbasis-Einheit des „Frame Grabbers“ sind in *Bild 3* dargestellt, ebenso das Write-Enable-Signal für die statischen RAMs (/WRITE), das Flash-A/D-Bus-Ausgangs-Enable (/FLOE) und das Adreß-Zähler-Inkrement-Signal (INCADR).

Die A/D-Umsetzung mit PAL

Bei den meisten Videokameras ist das RS-170-Signal gewöhnlich an der Quelle wechselspannungsgekoppelt. Diese Tatsache stellt ein kleines Problem für den „Frame Grabber“ dar, weil der Gleichspannungsanteil mit dem Pegel des Videosignals schwankt. Vor einer A/D-Umsetzung muß also durch eine „Klemm“-Schaltung ein definierter, konstanter Bezugspunkt wiederhergestellt werden.

Ein Flash-A/D-Umsetzer besteht in der Regel aus einer Reihe von Linear-Komparatoren mit dazugehörigen Widerstandsnetzwerken zur Erzeugung der entsprechenden Referenzspannungen. Die eben erwähnte „Klemmung“ wird generell durch das Entladen eines Kopplungskondensators mittels eines Transistors während des Intervalls des horizontalen Rücklaufs erreicht. Für Applikationen mit etwas geringerer Auflösung vollbringt jedoch eine wesentlich einfachere PAL-Schaltung die gleiche Leistung. Der „Clamp/Flash-Umsetzer“ ist in *Bild 4* dargestellt. Er besteht im wesentlichen aus einem

Tabelle 3

```
TITLE           FRAME GRABBER
PATTERN         PALHB5.PDF
REVISION        A
AUTHOR          ALFIE GILBERT/WILLY VOLDAN
COMPANY         MONOLITHIC MEMORIES INC.
DATE            18.JULI 1985

CHIP            FRAME_GRABBER PAL64R32

INC /CLRADR MODE /PWRUP /FBRRD /FBRWE NC NC PL1 PS1 GND CLK1 /OE1 Q716MHZ
Q358MHZ Q179MHZ Q090MHZ /FLOE /WRITE INCADR INCDLY /CS0 /CS1 /CS2 /CS3
/RAMOE /RAMWE NC NC /OE2 CLK2 VCC PS2 PL2 NC NC NC NC NC NC NC NC NC NC
NC NC NC NC NC NC PL3 PS3 GND CLK3 /OE3 A11 A10 A12 A9 A13 A8 A14 CARRY
A4 A3 A5 A2 A6 A1 A7 A0 /OE4 CLK4 VCC PS4 PL4 NC NC NC NC NC NC NC NC

EQUATIONS
;
;              MODUL FÜR DIE ZEITBASIS
;
; Diese PAL-Schaltung ist ein synchroner 4-Bit Zähler. Die Ausgänge
; dieses Zählers werden intern zur Erzeugung der richtigen Phase ver-
; schiedener Signale zur Adreß- oder Datensteuerung benutzt. Der Zeit-
; basis-Zähler arbeitet unabhängig (d.h. er braucht kein Zählerfrei-
; gabesignal), er kann jedoch immer synchron zurückgesetzt werden wenn
; an das Eingangssignal /PWRUP ein "aktiv-low" angelegt wird.
; Der Oszillator des Systems mit einer Frequenz von 14,31818MHz taktet
; das Modul der Zeitbasis.
;
;
Q716MHZ  :=/Q716MHZ * /PWRUP                                ;TOGGLE OUTPUT
                                                            ;RESET  OUTPUT

Q358MHZ  := Q716MHZ * /Q358MHZ * /PWRUP                     ;TOGGLE OUTPUT
           +/Q716MHZ *  Q358MHZ * /PWRUP                    ;TOGGLE OUTPUT
                                                            ;RESET  OUTPUT

Q179MHZ  := Q716MHZ * Q358MHZ * /Q179MHZ * /PWRUP           ;TOGGLE OUTPUT
           +/Q716MHZ          *  Q179MHZ * /PWRUP           ;TOGGLE OUTPUT
           +          /Q358MHZ *  Q179MHZ * /PWRUP          ;TOGGLE OUTPUT
                                                            ;RESET  OUTPUT

Q090MHZ  := Q716MHZ * Q358MHZ * Q179MHZ * /Q090MHZ * /PWRUP ;TOGGLE OUTPUT
           +/Q716MHZ                    *  Q090MHZ * /PWRUP ;TOGGLE OUTPUT
           +          /Q358MHZ          *  Q090MHZ * /PWRUP ;TOGGLE OUTPUT
           +                   /Q179MHZ *  Q090MHZ * /PWRUP ;TOGGLE OUTPUT
                                                            ;RESET  OUTPUT
;
;          MODUL FÜR DIE RAM-ADRESSGENERIERUNG
;
; Diese PAL-Schaltung ist ein 15-Bit Synchron-Zähler. Die Ausgänge dieses
; Zählers adressieren den RAM-Speicher des Bildraster-Speichers.
; Der Speicher hat eine Kapazität von 32K Byte. Da vier 8K-Byte RAMs ver-
; wendet werden sind die Ausgänge A14 und A13 nicht direkt mit dem Speicher
; verbunden, sie dienen jedoch als Eingänge für das "RAM MEMORY DECODER
; MODULE".
; Ein gespeichertes "LOOK-AHEAD CARRY"-Bit (CARRY) verbindet das Low- und
; High-Byte der Adresse. Der Adreßzähler wird durch Anlegen von "aktiv-
; high" am Signal INCADR freigegeben. Durch das Anlegen von "aktiv-low" am
; Eingangssignal /CLRADR kann der Zähler auf 0 zurückgesetzt werden.
; Der Zähler wird vom gleichen 14,3MHz Oszillator wie der Zeitbasis-
; Generator getaktet.
;
A0   :=/A0 *  INCADR * /CLRADR                              ;TOGGLE OUTPUT
      + A0 * /INCADR * /CLRADR                              ;HOLD   OUTPUT
                                                            ;RESET  OUTPUT

A1   := A0 * /A1 *  INCADR * /CLRADR                        ;TOGGLE OUTPUT
      +/A0 *  A1 *  INCADR * /CLRADR                        ;TOGGLE OUTPUT
      +       A1 * /INCADR * /CLRADR                        ;HOLD   OUTPUT
                                                            ;RESET  OUTPUT

A2   := A0 *  A1 * /A2 *  INCADR * /CLRADR                  ;TOGGLE OUTPUT
      +/A0       *  A2 *  INCADR * /CLRADR                  ;TOGGLE OUTPUT
      +      /A1 *  A2 *  INCADR * /CLRADR                  ;TOGGLE OUTPUT
      +             A2 * /INCADR * /CLRADR                  ;HOLD   OUTPUT
                                                            ;RESET  OUTPUT

A3   := A0 *  A1 *  A2 * /A3 *  INCADR * /CLRADR            ;TOGGLE OUTPUT
      +/A0             *  A3 *  INCADR * /CLRADR            ;TOGGLE OUTPUT
      +      /A1       *  A3 *  INCADR * /CLRADR            ;TOGGLE OUTPUT
      +            /A2 *  A3 *  INCADR * /CLRADR            ;TOGGLE OUTPUT
      +                   A3 * /INCADR * /CLRADR            ;HOLD   OUTPUT
                                                            ;RESET  OUTPUT

A4   := A0 *  A1 *  A2 *  A3 * /A4 *  INCADR * /CLRADR      ;TOGGLE OUTPUT
      +/A0                   *  A4 *  INCADR * /CLRADR      ;TOGGLE OUTPUT
      +      /A1             *  A4 *  INCADR * /CLRADR      ;TOGGLE OUTPUT
      +            /A2       *  A4 *  INCADR * /CLRADR      ;TOGGLE OUTPUT
      +                  /A3 *  A4 *  INCADR * /CLRADR      ;TOGGLE OUTPUT
      +                         A4 * /INCADR * /CLRADR      ;HOLD   OUTPUT
                                                            ;RESET  OUTPUT

A5   := A0 *  A1 *  A2 *  A3 * A4 * /A5 *
                                      INCADR * /CLRADR      ;TOGGLE OUTPUT
      +/A0                   * A5 *   INCADR * /CLRADR      ;TOGGLE OUTPUT
      +      /A1             * A5 *   INCADR * /CLRADR      ;TOGGLE OUTPUT
      +            /A2       * A5 *   INCADR * /CLRADR      ;TOGGLE OUTPUT
      +                  /A3 * A5 *   INCADR * /CLRADR      ;TOGGLE OUTPUT
      +                  /A4 * A5 *   INCADR * /CLRADR      ;TOGGLE OUTPUT
      +                        A5 *  /INCADR * /CLRADR      ;HOLD   OUTPUT
                                                            ;RESET  OUTPUT

A6   := A0 *  A1 *  A2 *  A3 * A4 *  A5 * /A6 *
                                     INCADR * /CLRADR       ;TOGGLE OUTPUT
      +/A0                   * A6 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +      /A1             * A6 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +             /A2      * A6 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +                  /A3 * A6 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +                  /A4 * A6 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +                  /A5 * A6 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +                        A6 * /INCADR * /CLRADR       ;HOLD   OUTPUT
                                                            ;RESET  OUTPUT

A7   := A0 *  A1 *  A2 *  A3 * A4 *  A5 *  A6 * /A7 *
                                     INCADR * /CLRADR       ;TOGGLE OUTPUT
      +/A0                   * A7 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +      /A1             * A7 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +            /A2       * A7 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +                  /A3 * A7 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +                  /A4 * A7 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +                  /A5 * A7 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +                  /A6 * A7 *  INCADR * /CLRADR       ;TOGGLE OUTPUT
      +                        A7 * /INCADR * /CLRADR       ;HOLD   OUTPUT
                                                            ;RESET  OUTPUT

CARRY:=/A0 * A1 * A2 * A3 * A4 * A5 * A6 * A7 *
                                     INCADR * /CLRADR       ;TOGGLE CARRY
      +                 CARRY     */INCADR * /CLRADR        ;HOLD   CARRY
                                                            ;OTHERWISE
                                                            ;RESET  CARRY

A8   :=/A8 *  CARRY *  INCADR * /CLRADR                     ;TOGGLE OUTPUT
      + A8 * /CARRY            * /CLRADR                    ;HOLD   OUTPUT
      + A8           * /INCADR * /CLRADR                    ;HOLD   OUTPUT


A9   := A8 * /A9 *  CARRY *  INCADR * /CLRADR               ;TOGGLE OUTPUT
      +/A8 *  A9 *  CARRY *  INCADR * /CLRADR               ;TOGGLE OUTPUT
      +       A9 * /CARRY            * /CLRADR              ;HOLD   OUTPUT
      +       A9           * /INCADR * /CLRADR              ;HOLD   OUTPUT

 A10  := A8 *  A9 * /A10 *  CARRY *  INCADR * /CLRADR       ;TOGGLE OUTPUT
       +/A8       *  A10 *  CARRY *  INCADR * /CLRADR       ;TOGGLE OUTPUT
       +      /A9 *  A10 *  CARRY *  INCADR * /CLRADR       ;TOGGLE OUTPUT
       +             A10 * /CARRY            * /CLRADR      ;HOLD   OUTPUT
       +             A10           * /INCADR * /CLRADR      ;HOLD   OUTPUT

  A11  := A8 *  A9 * A10 * /A11 *
                           CARRY *  INCADR * /CLRADR        ;TOGGLE OUTPUT
        +      /A8 * A11 *  CARRY *  INCADR * /CLRADR       ;TOGGLE OUTPUT
        +      /A9 * A11 *  CARRY *  INCADR * /CLRADR       ;TOGGLE OUTPUT
        +     /A10 * A11 *  CARRY *  INCADR * /CLRADR       ;TOGGLE OUTPUT
        +            A11 * /CARRY            * /CLRADR      ;HOLD   OUTPUT
        +            A11           * /INCADR * /CLRADR      ;HOLD   OUTPUT

A12 := A8 *  A9 * A10 *  A11 * /A12 *
                       CARRY *  INCADR * /CLRADR            ;TOGGLE OUTPUT
     +      /A8 * A12 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +      /A9 * A12 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +     /A10 * A12 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +     /A11 * A12 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +            A12 * /CARRY            * /CLRADR         ;HOLD   OUTPUT
     +            A12           * /INCADR * /CLRADR         ;HOLD   OUTPUT
```

PAL16R8 und einigen passiven Bauelementen und arbeitet überaus zufriedenstellend. Dieses Schema kann einfach auf 3 Bit Auflösung, bei der acht Schwellwerte codiert werden müssen, ausgebaut werden, wenn Widerstände mit 1 % Toleranz in der Kette verwendet werden. Bei noch höheren Ansprüchen lassen sich dem PAL noch zusätzlich Komparatoren vorschalten.

Die Arbeitsweise des „Frame Grabbers"

Die Gesamtschaltung des „Frame Grabbers" zeigt *Bild 5*. Drei PAL-Bauelemente sind das Herz des Systems. Das MegaPAL PAL64R32 handhabt den Großteil der Zeitabläufe, der Adressierung und der Steuerlogik. Das PAL16R8 ist für die A/D-Umsetzung und die Schnittstelle des internen Busses zuständig, das PAL20RA10 dient zur Herstellung der Schnittstelle zwischen dem internen Bus und dem I/O-Bus des PC. Zur Vervollständigung des Systems bilden vier statische RAMs mit je 8 KByte den Pufferspeicher, und ein Octal-Transceiver (74LS245) stellt genügend Treiberleistung für den I/O-Kanal bereit.

Der „Frame Grabber" hat zwei Betriebsarten: Ist das MODE-Steuerbit auf logisch „High", so befindet sich das System im Zustand der Bilderfassung. Das Video-Eingangssignal wird fortlaufend digitalisiert und im RAM abgespeichert. Im Lesezustand (dabei ist das MODE-Bit

```
A13  := A8 *  A9 * A10 *  A11 * A12 * /A13 *
                          CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +      /A8 * A13 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +      /A9 * A13 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +     /A10 * A13 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +     /A11 * A13 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +     /A12 * A13 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +            A13 * /CARRY           * /CLRADR          ;HOLD   OUTPUT
     +            A13          */ INCADR * /CLRADR          ;HOLD   OUTPUT

A14  := A8 *  A9 * A10 *  A11 * A12 * A13 * /A14 *
                          CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +      /A8 * A14 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +      /A9 * A14 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +     /A10 * A14 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +     /A11 * A14 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +     /A12 * A14 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +     /A13 * A14 *  CARRY *  INCADR * /CLRADR          ;TOGGLE OUTPUT
     +            A14 * /CARRY            * /CLRADR          ;HOLD   OUTPUT
     +            A14           * /INCADR * /CLRADR          ;HOLD   OUTPUT

;
;          MODUL FÜR DIE DEKODIERUNG DES RAM SPEICHERS
;
; Diese PAL-Schaltung ein einfacher kombinatorischer 2-zu-4 Zeilendekoder.
; Die Ausgänge (/CS0,/CS1,/CS2,/CS3) werden mit den "aktiv-low" Eingängen
; des "Chip-Select" der vier statischen RAMs verbunden.
; Da die Dekoder-Ausgänge "aktiv-low" sind wird dieses Modul in negativer
; Logik aufgebaut. Man beachte, daß der logische Sinn der folgenden
; Ausgangsvariablen entgegengesetzt den Bezeichnungen der Pin-Liste ist,
; z.B. CS0 gegen /CS0. Dies ist grundlegend für negative Logik.
;
CS0   =/A14 * /A13      ;OUTPUT ASSERTS ON BINARY COUNT OF 0 0

CS1   =/A14 *  A13      ;OUTPUT ASSERTS ON BINARY COUNT OF 0 1

CS2   = A14 * /A13      ;OUTPUT ASSERTS ON BINARY COUNT OF 1 0

CS3   = A14 *  A13      ;OUTPUT ASSERTS ON BINARY COUNT OF 1 1


;
;           MODUL FÜR DIE STEUERUNG DES RAM-SPEICHERS
;
; Diese Baugruppe erzeugt einige verschiedenartige Steuer-Signale,
; die im Folgenden einzeln beschrieben werden.
; Einige der Ausgänge sind kombinatorisch, andere "registered".
;
; Dieses Signal (/WRITE) ist ein negativer Impuls mit 70ns, der die
; RAMs im Aufnahmezustand des "FRAME-GRABBER" beschreibt. Dieser
; Ausgang stammt vom Zeitbasis-Generator und ist in negativer Logik
; ausgeführt.
;
WRITE := Q716MHZ * /Q358MHZ * Q179MHZ * Q090MHZ * MODE * /PWRUP

; Diese Gleichung ist ein 2-zu-1 Daten-Multiplexer. Das Ausgangs-
; signal (/RAMWE) kann als Schreibfreigabe für die statischen RAMs
; verwendet werden.
; Dieser Ausgang ist indirekt abgeleitet entweder vom Zeitbasis-
; Generator (/WRITE) oder dem Steuerbus (/FBRWE) des PC.
; Damit schreibt der PC durch einen "I/O-WRITE" Befehl in den
; "FRAME-GRABBER".
;
/RAMWE =/WRITE *  MODE
       +/FBRWE * /MODE


; Diese Gleichung ist ebenfalls ein 2-zu-1 Mux, wiederum gesteuert
; vom "MODE-Signal". Dieses Signal gibt den Adreßzähler frei, sodaß
; er hochzählen kann (zur Erinnerung: der Adreßzähler wird vom
; 14,3MHz Systemtakt getaktet). Der Ausgang ist vom Zeitbasis-
; Generator abgeleitet wenn der "FRAME-GRABBER" im Aufnahmezustand
; ist. Oder er wird abgeleitet von der ansteigenden Flanke des Ein-
; gangs-Signals INC wenn die Einheit im Lesezustand arbeitet. INCADR
; ist ein Impuls positiver Polarität mit einer Zeitdauer, die
; einem Systemtakt von 70ns entspricht.
;
INCADR :=/Q716MHZ * Q358MHZ * Q179MHZ * Q090MHZ * MODE * /PWRUP
        +/INCDLY  * INC     */MODE    */PWRUP

; Diese Logik synchronisiert das Eingangssignal INC mit dem 14,3MHz
; Systemtakt. Man beachte, daß die Zustände von INCDLY auf "Low"
; und INC auf "High" einen Synchrondetektor der ansteigenden Flanke
; des Eingangssignals INC darstellen (siehe auch das zweite Produkt
; in der logischen Gleichung für INCADR).
;
INCDLY := INC


;
;             MODUL FÜR DIE STEUERUNG DES INTERNEN BUSSES
;
; Diese PAL-Schaltung erzeugt Signale, die die Three-State Ausgänge
; des "FLASH-A/D WANDLERS" und der statischen RAMs freigeben.
; Das "FLASH"-Ausgangsfreigabesignal (/FLOE) ist hergeleitet vom Zeit-
; basisgenerator und wird von der Einheit im Aufnahmezustand ange-
; sprochen. Das Speicher-Ausgangs-Freigabesignal (/RAMOE) ist vom
; Steuerbus des PC abgeleitet wenn der "FRAME-BUFFER" im Lesezustand
; ist.
;
/RAMOE =/FBRRD * /MODE
       + MODE

FLOE  :=/Q716MHZ *  Q358MHZ * Q179MHZ * Q090MHZ * MODE * /PWRUP
       + Q716MHZ * /Q358MHZ * Q179MHZ * Q090MHZ * MODE * /PWRUP


SIMULATION

TRACE_ON    CLK1 CLK2 CLK3 CLK4                       ;CLOCK SIGNALS
           /PWRUP MODE FLOE WRITE INCADR              ;CONTROL SIGNALS
            Q716MHZ Q358MHZ Q179MHZ Q090MHZ           ;TIME BASE SIGNALS
            A0 A1 A2 A3 A4 A5 A6 A7 A8 A9             ;ADDRESS SIGNALS

SETF OE1 OE2 OE3 OE4                                  ;ENABLE OUTPUTS
     PS1 PS2 PS3 PS4                                  ;UNASSERT SET
     PL1 PL2 PL3 PL4                                  ;UNASSERT PRELOAD

SETF PWRUP MODE                                       ;CLEAR TIME BASE
CLOCKF CLK1 CLK2 CLK3 CLK4

SETF /PWRUP                                           ;EXERCISE TIME BASE
FOR I:=1 TO 17 DO
 BEGIN
  CLOCKF CLK1 CLK3 CLK4
 END

SETF CLRADR                                           ;CLEAR ADDRESS COUNTER
CLOCKF CLK1 CLK3 CLK4

SETF /CLRADR                                          ;TOGGLE A1 AND A0
FOR I:=1 TO 32 DO
 BEGIN
  CLOCKF CLK1 CLK3 CLK4
 END

SETF  A0  A1 /A2 /A3 /A4
     /A5 /A6 /A7 /A8 /A9
     /A10 /A11 /A12 /A13 /A14                          ;TOGGLE A2

FOR I:=1 TO 187 DO
 BEGIN
  CLOCKF CLK1 CLK3 CLK4
  IF I=17  THEN BEGIN SETF A2  END                     ;TOGGLE A3
  IF I=17  THEN BEGIN SETF A3  END                     ;TOGGLE A4
  IF I=17  THEN BEGIN SETF A4  END                     ;TOGGLE A5
  IF I=17  THEN BEGIN SETF A5  END                     ;TOGGLE A6
  IF I=17  THEN BEGIN SETF A6  END                     ;TOGGLE A7
  IF I=17  THEN BEGIN SETF A7  END                     ;TOGGLE A8
  IF I=17  THEN BEGIN SETF A8  END                     ;TOGGLE A9
  IF I=17  THEN BEGIN SETF A9  END                     ;TOGGLE A10
  IF I=17  THEN BEGIN SETF A10 END                     ;TOGGLE A11
  IF I=17  THEN BEGIN SETF A11 END                     ;TOGGLE A12

 END


SETF /A14 /A13                                        ;EXERCISE RAM DECODER
SETF /A14  A13
SETF  A14 /A13
SETF  A14  A13
```

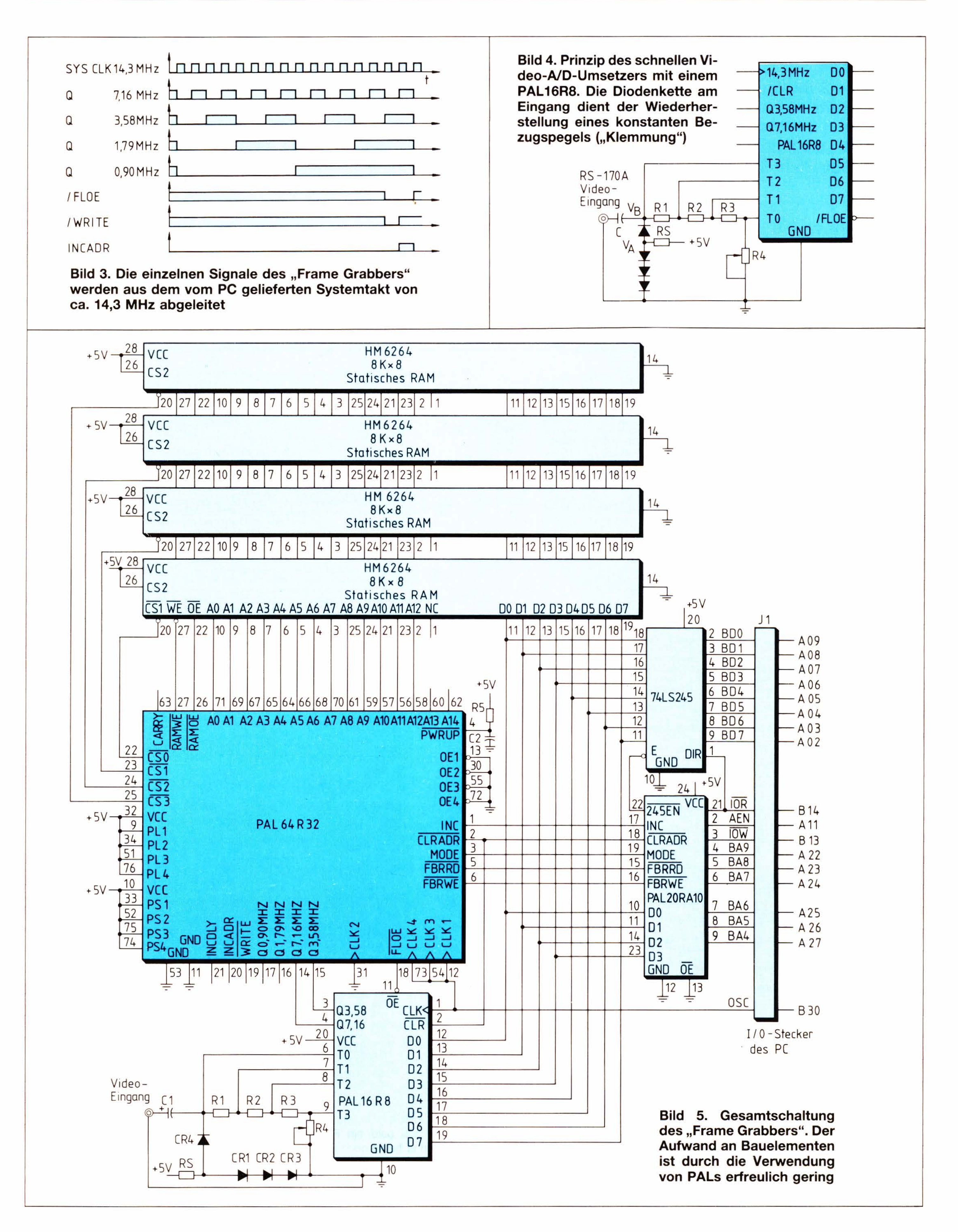


Bild 3. Die einzelnen Signale des „Frame Grabbers" werden aus dem vom PC gelieferten Systemtakt von ca. 14,3 MHz abgeleitet

Bild 4. Prinzip des schnellen Video-A/D-Umsetzers mit einem PAL16R8. Die Diodenkette am Eingang dient der Wiederherstellung eines konstanten Bezugspegels („Klemmung")

Bild 5. Gesamtschaltung des „Frame Grabbers". Der Aufwand an Bauelementen ist durch die Verwendung von PALs erfreulich gering

„Low“) übernimmt der Ein-/Ausgabekanal des PC die Speichersteuerung. Der Adreßzähler für die Ein-/Ausgabe-Zuordnung kann ab Adresse 100H zugreifen.



Bild 6. Nicht gerade „High Videlity“, aber immerhin ein gut erkennbares Bild liefert der „Frame Grabber“ auf dem Schirm eines PC. Rechts der über das Ergebnis offensichtlich erfreute Autor des Originalbeitrags (A. Gilbert). Der etwas nachdenklich gestimmte Herr neben ihm ist John Birkner, der Vater der PAL-Idee bei MMI

Einsatzgebiete der Rasterbildspeicherung

Das Konzept eines Rasterbildspeichers ist einfach erläutert: Er speichert die ankommende Video-Information in einem RAM-Feld für spätere Verwendung. Das digitalisierte, abgespeicherte Bild wird häufig mittels digitaler Signalverarbeitungsverfahren bearbeitet. Diese Verfahren können auf Hardware-Konfigurationen oder Software-Grundlage basieren. Die digitale Signalverarbeitung, die in die Hardware eingearbeitet ist, ist ein sehr schnelles Verfahren (bis zur „Echtzeit“), jedoch kostspielig. Die Signalverarbeitung mittels Software-Algorithmen ist langsamer, aber bei vielen Applikationen kosteneffizienter.

Die grundlegendste Art der digitalen Signalverarbeitung, die gewöhnlich bei Video ausgeführt wird, dient der Verbesserung der Bildqualität. Dies betrifft vor allem Videosignale, die von Rauschen gestört werden. Viele Anwendungsfälle gehen aber über die Bildverbesserung hinaus zu komplexeren Arten der Signalverarbeitung, z. B. der Mustererkennung. Meist ist dieser Prozeß ein Software-Algorithmus, der nicht in „Echtzeit“ abläuft. Dafür müssen digitale Rasterbilder in einem Speicher gepuffert werden. Die zunehmende Popularität von Video-Rasterbildspeichern wurde sowohl von der Anwendungsvielfalt als auch den Technologien beeinflußt. Die stetig sinkenden Preise für RAMs machten es möglich, den zur Speicherung von Videobildern mit akzeptabler Auflösung und damit Grauskala nötigen Speicherplatz vorzusehen. ASIC-Techniken, wie beispielsweise das in dieser Anwendung eingesetzte Mega-PAL, waren ebenfalls für ein rationelles Entwickeln der Adreß-, Steuer- und Zeitablauflogik verantwortlich. Diese technischen Fortschritte haben viele neue Video-Applikationen wirtschaftlich nutzbar gemacht. Auf dem industriellen Sektor sind hier zu nennen die Robotersteuerung, Qualitätskontrolle, Qualitätssicherung, Eingangskontrolle, Überwachungs- und Sicherheitssysteme und eine ganze Reihe weiterer Einsatzgebiete.

Die Zukunft für Video-Anwendungen kann sich durchaus in der Umgebung von Personal Computern abspielen. Die Verbindung einer optischen Einheit mit einem Personal Computer stellt gleichsam das Silizium-Gegenstück zu einem sehr bekannten organischen Rechenwerk dar – unsere Augen und das Gehirn. Es ist nur eine Frage der Zeit, wann PCs für uns lesen und Dinge erkennen können werden.

Literatur:

[1] *Gilbert, Alfie:* Video Frame Grabber. AN-148 im System Design Handbook, Second Edition. Monolithic Memories Inc., 1985.

[2] PAL (Programmable Array Logic) Handbook. Third Edition. Monolithic Memories Inc., 1983.

[3] *Mäusl, R.:* Fernsehtechnik. Von der Kamera zum Bildschirm. Pflaum Verlag, München, 1981.

Alfi Gilbert ist Mitarbeiter von MMI

Gerhard Kirschner ist gebürtiger Münchner. Hier studierte er auch Kommunikationswissenschaften. Während des Studiums begann er, „Marketing Communications“ bei Monolithic Memories GmbH in München zu betreiben, woraus bald ein Full-Time-Job wurde, den er heute noch gerne erfüllt.



Willibald Voldan

Einfache PAL-Schaltung zur Drehrichtungserkennung

Wegen ihrer Flexibilität, der hohen Verarbeitungsgeschwindigkeit und exakt gleicher Laufzeiten im Chip sind PAL-Bausteine u. a. gut für Shaft-Encoder-Schaltungen zur Positionserfassung beweglicher Maschinenteile geeignet.

Encoder-Prinzip

Beim inkrementalen Umsetzungsverfahren erzeugt die Änderung des Eingangswertes um eine Quantisierungseinheit jeweils einen Zählimpuls (Impulszählverfahren). Ändert sich der Meßwert in positiver oder negativer Richtung, so muß auch die Bewegungsrichtung durch die Art des Ausgangssignals kenntlich gemacht werden. Hierzu erzeugt der Inkrementalgeber in der Regel zwei Ausgangssignale.

Bild 1 zeigt den grundsätzlichen Aufbau eines Inkrementalgebers für drehende Bewegungen. Dieser „optische Encoder" besteht aus einer Scheibe mit mindestens zwei Reihen von lichtdurchlässigen und lichtundurchlässigen Segmenten, die zueinander um 90° versetzt sind (Strichgitter). Vor bzw. hinter dieser Scheibe befinden sich Lichtquellen bzw. lichtempfindliche Sensoren, die im Zusammenspiel mit den um eine Viertelperiode zueinander versetzten Feldern Signale erzeugen (*Bild 2*), die über die Phasenlage eine Drehrichtungserkennung ermöglichen.

Jede Änderung der Spannung über eine Periode stellt eine Änderung des Einheitssignals um eine Quantisierungseinheit dar. Die annähernd sinusförmigen Signale (Inkrementalsignale) sind allerdings zur direkten Ansteuerung logischer Schaltungen nicht geeignet; es ist eine Verstärkung und Pulsformung erforderlich.

Die Schaltschwellen lassen sich mit Hilfe eines Schmitt-Triggers auf ein Verhältnis von Impulsdauer zur Gesamtperiodendauer von 2:1 einstellen. Die Ausgangssignale werden dann einem Drehrichtungsdiskriminator zugeführt und entsprechend verknüpft (*Bild 3*).

Um Schwierigkeiten mit eventuellen Störimpulsen zu entgehen, verwendet man in der Praxis nur taktgesteuerte synchrone Schaltungen. Im *Bild 4* ist das Impulsdiagramm eines Shaft-Encoders dargestellt. Das Prinzip dieser Schaltung liegt in der Verzögerung zwischen den Taktimpulsen für die zwei Signale PHI0 und PHI90:

S1 = PHI0 clk S2 = PHI0 clk + 1
S3 = PHI90 clk S4 = PHI90 clk + 1

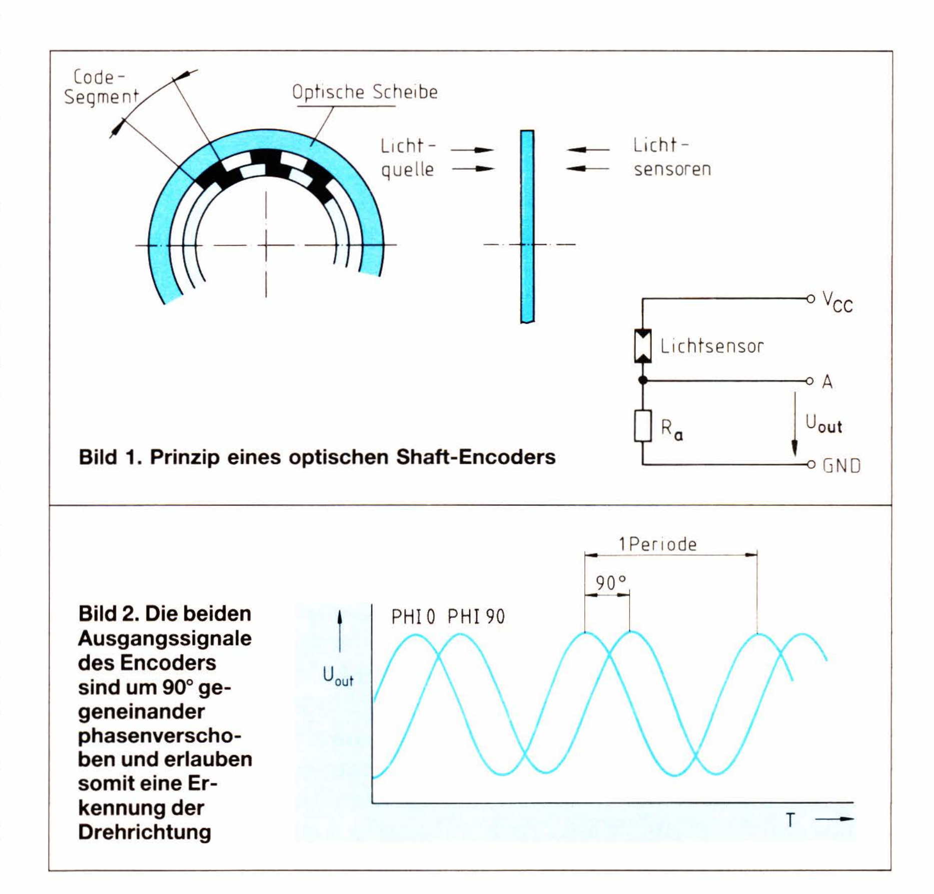


Bild 1. Prinzip eines optischen Shaft-Encoders

Bild 2. Die beiden Ausgangssignale des Encoders sind um 90° gegeneinander phasenverschoben und erlauben somit eine Erkennung der Drehrichtung

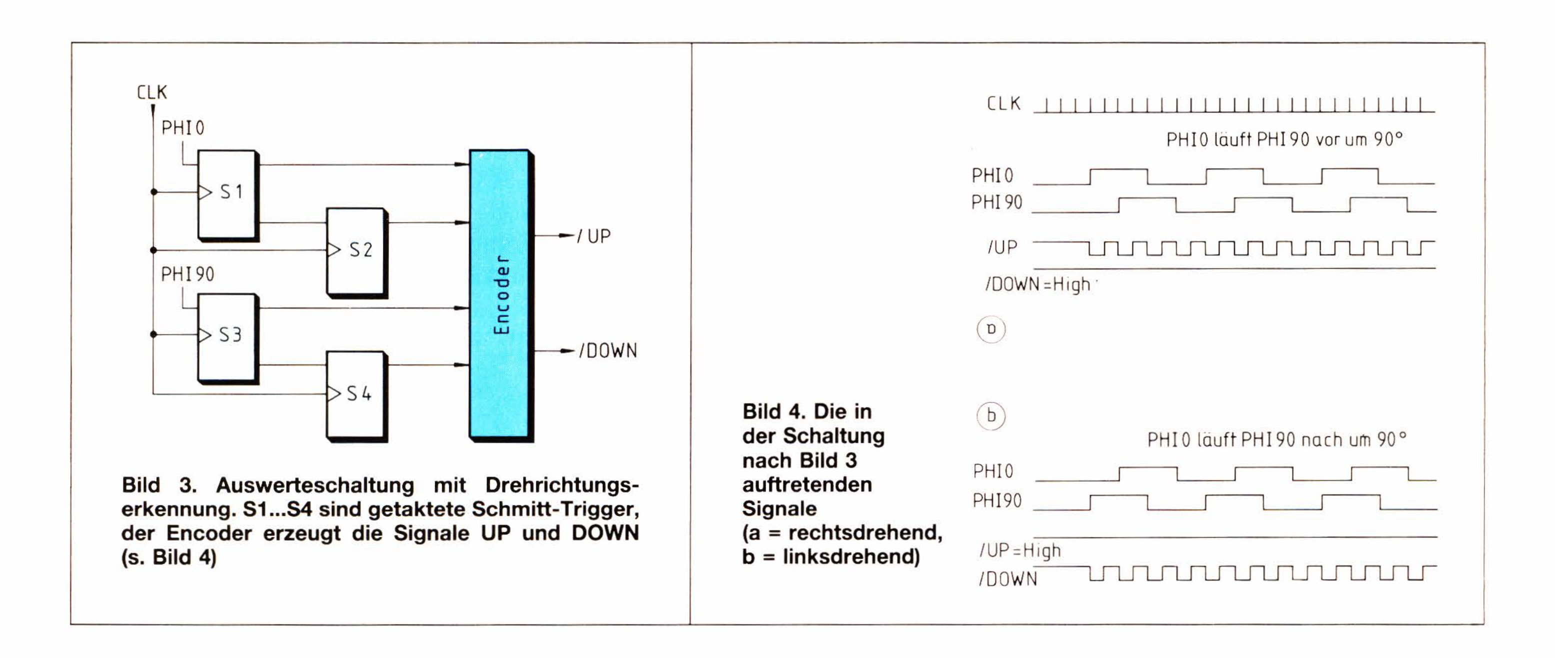


Bild 3. Auswerteschaltung mit Drehrichtungserkennung. S1...S4 sind getaktete Schmitt-Trigger, der Encoder erzeugt die Signale UP und DOWN (s. Bild 4)

Bild 4. Die in der Schaltung nach Bild 3 auftretenden Signale (a = rechtsdrehend, b = linksdrehend)

Ein Signalwechsel von PHI0 und PHI90 von L zu H (oder umgekehrt) bewirkt einen Wechsel der Daten in den Registern. Diese Änderung wird zur Auswertung decodiert. Die logischen Gleichungen für die Decoder-Schaltung lauten:
PHI0 läuft PHI90 vor:

S1 * S2 * S3 * /S4
/S1 * /S2 * /S3 * S4
S1 * /S2 * /S3 * /S4
/S1 * S2 * S3 * S4

PHI0 läuft PHI90 nach:

/S1 * /S2 * S3 * /S4
S1 * S2 * /S3 * S4
S1 * /S2 * S3 * S4
/S1 * S2 * /S3 * /S4

Das „*“-Zeichen symbolisiert die UND-Verknüpfung.

Nehmen wir an, daß die Scheibe sich im Uhrzeigersinn bewegt, dann erzeugt die Schaltung nur Impulse am Ausgang „/RECHTS“. Bewegt sich die Scheibe gegen den Uhrzeigersinn, so produziert der Ausgang „/LINKS“ Impulse. Der jeweils andere Ausgang wird bei dieser Schaltung auf logisch „High“ gehalten.

Um sicherzustellen, daß kein Pegelwechsel verloren geht, sollte die Taktfrequenz dieser synchronen Schaltung mindestens $8 \times N \times s$ sein.
N = Anzahl der Impulse des Encoders pro Umdrehung.
s = Maximale Geschwindigkeit in Umdrehungen pro Sekunde.

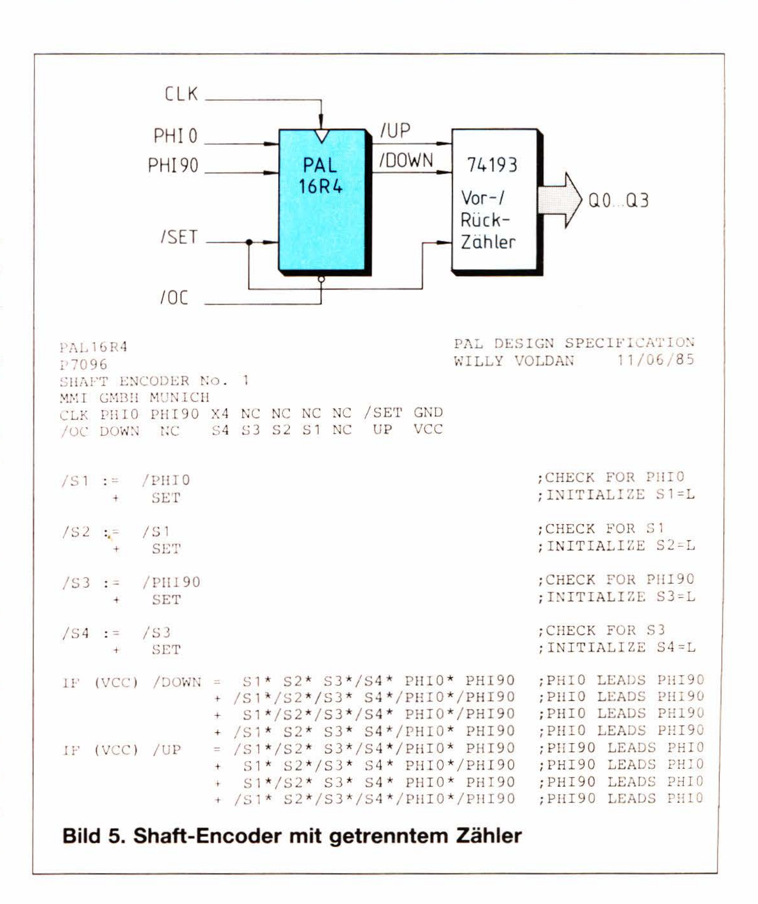
In der Praxis sind Taktfrequenzen im Bereich von 1 MHz am gebräuchlichsten.

Diese beschriebenen synchronen Shaft-Encoder-Schaltungen sind relativ unempfindlich gegen Phasen-Fehler des Encoders. Sie verwenden keine Mono-Flops und ermöglichen es, unerlaubte Übergangszustände der Schaltung zu erkennen. Eine Eigenheit dieser Schaltung ist, daß Störungen auf den Eingangsleitungen zwischen den Taktimpulsen ignoriert werden.

Shaft-Encoder mit PAL

Je nach Programmierung und PAL-Typ lassen sich die verschiedensten und der jeweiligen Applikation am besten angepaßten Shaft-Encoder-Schaltungen realisieren.

Eine grundlegende Schaltung zur Drehrichtungserkennung ergibt sich aus den logischen Gleichungen des PALASM-Programmes in *Bild 5*. Dabei werden dem PAL-Baustein 16R4 über die Anschlüsse PHI0 und




```
PAL16R4                                  PAL DESIGN SPECIFICATION
P7096                                    WILLY VOLDAN     11/06/85
SHAFT ENCODER No. 1
MMI GMBH MUNICH
CLK PHI0 PHI90 X4 NC NC NC NC /SET GND
/OC DOWN  NC   S4 S3 S2 S1 NC  UP  VCC

/S1 :=  /PHI0                                 ;CHECK FOR PHI0
     +   SET                                  ;INITIALIZE S1=L

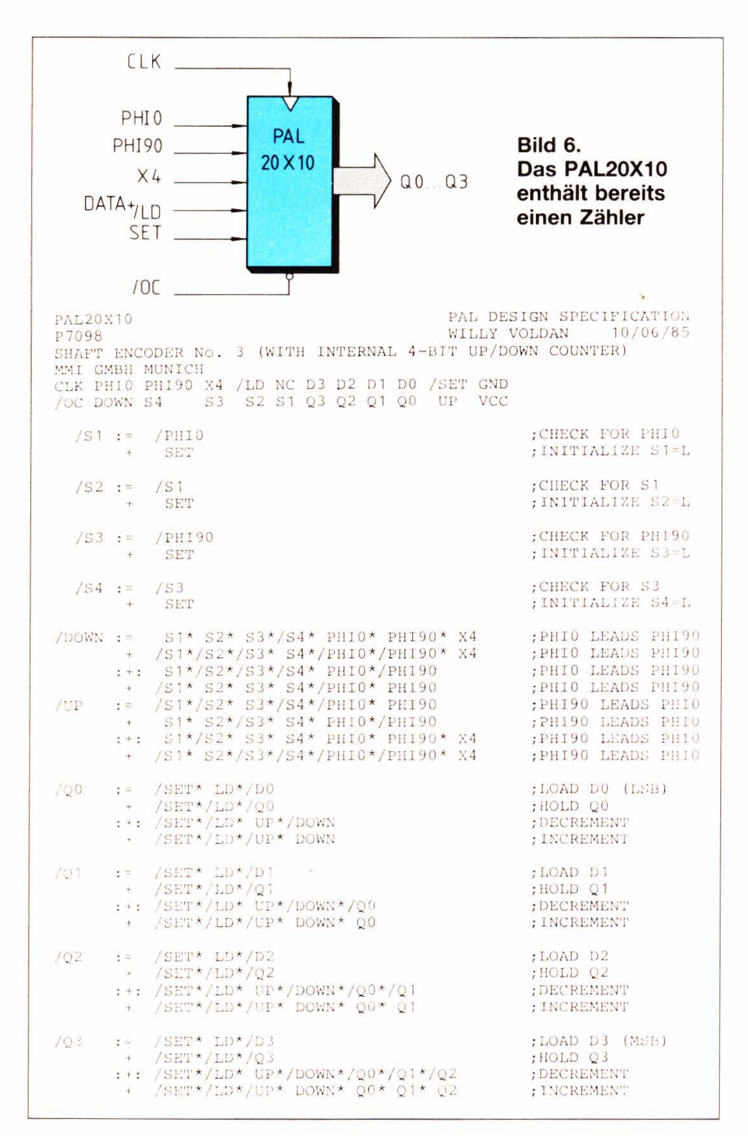
/S2 :=  /S1                                   ;CHECK FOR S1
     +   SET                                  ;INITIALIZE S2=L

/S3 :=  /PHI90                                ;CHECK FOR PHI90
     +   SET                                  ;INITIALIZE S3=L

/S4 :=  /S3                                   ;CHECK FOR S3
     +   SET                                  ;INITIALIZE S4=L

IF (VCC) /DOWN =  S1* S2* S3*/S4* PHI0* PHI90 ;PHI0 LEADS PHI90
               + /S1*/S2*/S3* S4*/PHI0*/PHI90 ;PHI0 LEADS PHI90
               +  S1*/S2*/S3*/S4* PHI0*/PHI90 ;PHI0 LEADS PHI90
               + /S1* S2* S3* S4*/PHI0* PHI90 ;PHI0 LEADS PHI90
IF (VCC) /UP   = /S1*/S2* S3*/S4*/PHI0* PHI90 ;PHI90 LEADS PHI0
               +  S1* S2*/S3* S4* PHI0*/PHI90 ;PHI90 LEADS PHI0
               +  S1*/S2* S3* S4* PHI0* PHI90 ;PHI90 LEADS PHI0
               + /S1* S2*/S3*/S4*/PHI0*/PHI90 ;PHI90 LEADS PHI0
```

Bild 5. Shaft-Encoder mit getrenntem Zähler




```
PAL20X10                                          PAL DESIGN SPECIFICATION
P7098                                             WILLY VOLDAN     10/06/85
SHAFT ENCODER No. 3 (WITH INTERNAL 4-BIT UP/DOWN COUNTER)
MMI GMBH MUNICH
CLK PHI0 PHI90 X4 /LD NC D3 D2 D1 D0 /SET GND
/OC DOWN S4       S3  S2 S1 Q3 Q2 Q1 Q0  UP  VCC

  /S1 :=  /PHI0                                  ;CHECK FOR PHI0
       +   SET                                   ;INITIALIZE S1=L

  /S2 :=  /S1                                    ;CHECK FOR S1
       +   SET                                   ;INITIALIZE S2=L

  /S3 :=  /PHI90                                 ;CHECK FOR PHI90
       +   SET                                   ;INITIALIZE S3=L

  /S4 :=  /S3                                    ;CHECK FOR S3
       +   SET                                   ;INITIALIZE S4=L

/DOWN :=   S1* S2* S3*/S4* PHI0* PHI90* X4       ;PHI0 LEADS PHI90
       +  /S1*/S2*/S3* S4*/PHI0*/PHI90* X4       ;PHI0 LEADS PHI90
      :+:  S1*/S2*/S3*/S4* PHI0*/PHI90           ;PHI0 LEADS PHI90
       +  /S1* S2* S3* S4*/PHI0* PHI90           ;PHI0 LEADS PHI90
/UP   :=  /S1*/S2* S3*/S4*/PHI0* PHI90           ;PHI90 LEADS PHI0
       +   S1* S2*/S3* S4* PHI0*/PHI90           ;PHI90 LEADS PHI0
      :+:  S1*/S2* S3* S4* PHI0* PHI90* X4       ;PHI90 LEADS PHI0
       +  /S1* S2*/S3*/S4*/PHI0*/PHI90* X4       ;PHI90 LEADS PHI0

/Q0   :=  /SET* LD*/D0                           ;LOAD D0 (LSB)
       +  /SET*/LD*/Q0                           ;HOLD Q0
      :+: /SET*/LD* UP*/DOWN                     ;DECREMENT
       +  /SET*/LD*/UP* DOWN                     ;INCREMENT

/Q1   :=  /SET* LD*/D1                           ;LOAD D1
       +  /SET*/LD*/Q1                           ;HOLD Q1
      :+: /SET*/LD* UP*/DOWN*/Q0                 ;DECREMENT
       +  /SET*/LD*/UP* DOWN* Q0                 ;INCREMENT

/Q2   :=  /SET* LD*/D2                           ;LOAD D2
       +  /SET*/LD*/Q2                           ;HOLD Q2
      :+: /SET*/LD* UP*/DOWN*/Q0*/Q1             ;DECREMENT
       +  /SET*/LD*/UP* DOWN* Q0* Q1             ;INCREMENT

/Q3   :=  /SET* LD*/D3                           ;LOAD D3 (MSB)
       +  /SET*/LD*/Q3                           ;HOLD Q3
      :+: /SET*/LD* UP*/DOWN*/Q0*/Q1*/Q2         ;DECREMENT
       +  /SET*/LD*/UP* DOWN* Q0* Q1* Q2         ;INCREMENT
```

Bild 6. Das PAL20X10 enthält bereits einen Zähler

PHI90 die beiden um 90° versetzten Inkrementalsignale zugeführt. Der Eingang X4 dient zur Umschaltung für eine Vervierfachung der Ausgangs-Signale.

Entsprechend der Drehrichtung der Encoder-Scheibe taktet der Ausgang „/DOWN" oder „/UP" synchron zum Systemtakt. Der jeweils andere Ausgang liegt auf logisch „High". Diese Konfiguration produziert somit die nötigen Steuersignale zum Anschluß eines Zählers der Type 74193.

Je nach der Drehrichtung zählt der Zähler vorwärts oder rückwärts und liefert damit einer CPU jeweils die genaue Stellung einer Maschine.

Eine komplette Lösung bietet die in *Bild 6* gezeigte PAL-Schaltung mit bereits integriertem Zähler, der die jeweilige Position des zu erfassenden mechanischen Teiles in binär codierter Form anzeigt.

Die beiden vom Inkrementalgeber gelieferten digitalen Signale PHI0 und PHI90 werden dem PAL20X10 zugeführt. Läuft das Signal PHI0 dem Signal PHI90 vor, zählt der im PAL vorgesehene Zähler vorwärts und umgekehrt.

Ein „SET"-Eingang ermöglicht die Rücksetzung der Zählerausgänge; das ist z. B. nach einem Stromausfall oder nach Einschalten des Gerätes wichtig. Mit dem Steuereingang „X4" kann die Frequenz des Zählers umgeschaltet werden.

Literatur

Voldan, W.: Shaft Encoder. System Design Handbook, MMI, Santa Clara, CA., 1985, S. 13-7 ff.

Arbeitshilfe

Die ELEKTRONIK ist für Sie eine echte Arbeitshilfe, wenn Sie sich mit der Entwicklung und industriellen Anwendung elektronischer Schaltungen und Baugruppen von Geräten und Systemen befassen.



Die ELEKTRONIK liefert Ihnen Informationen zur Konzeption, Projektion, Entwicklung und über die Einführung elektronischer Systeme in die Fertigung.

Die ELEKTRONIK informiert Sie über Bauelemente:
● Speicher aller Art ● Mikroprozessoren ● Logik-Bausteine ● Interface-Bausteine ● Signalprozessoren ● Operationsverstärker ● A/D- und D/A-Umsetzer ● Leistungshalbleiter ● Passive Bauelemente ● Elektromechanische Bauelemente ● Optoelektronische Bauelemente ● Sensoren und Aktuatoren.

Die ELEKTRONIK informiert Sie über Meß- und Prüftechnik:
Die ELEKTRONIK beobachtet die Marktentwicklung, beschreibt technische Grundlagen, gibt Spezialinformationen und macht so den Markt für den Interessenten überschaubar.

Die ELEKTRONIK informiert Sie über Automatisierung:
Besonders intensiv behandelt wird das Gebiet der Klein- und Mittelserienfertigung. Vielfältige Beispiele zeigen, was Industrieroboter in der Automatisierung zu leisten vermögen.

Die ELEKTRONIK informiert Sie über Kommunikationstechnik:
Einige Stichworte:
● Technologien der elektronischen Kommunikation ● Bauelemente für die Telekommunikation ● Übertragungsverfahren ● Meßtechnik ● Normung ● Internationale Zusammenarbeit ● Neue Medien ● Pläne der Fernmeldebehörden.

Die ELEKTRONIK informiert Sie über Schaltungspraxis:
Hier werden erprobte und aktuelle Schaltungen, Entwurfs- und Berechnungshilfen, Meßideen und Anwendungsbeispiele aus dem professionellen und dem industriellen Bereich präsentiert. Beispiele:
● Allgemeine Digitaltechnik ● Interface-Schaltungen ● Signalerzeugung ● Meßtechnik ● Meßwerterfassung und Verarbeitung ● Stromversorgung ● Spezialschaltungen ● µC-Praxis ● Applikationen.

Die ELEKTRONIK informiert Sie über Mikrocomputer:
● Tischcomputer ● Mikroprozessor-CPUs (Bauelemente und Platinen) ● Einchipcomputer ● Mikroprozessor-Peripherie (Bauelemente und Geräte) ● Speicher (Systeme und Bauelemente) ● Software (Systemebene, Anwenderebene, Sprachen) ● Entwicklungssysteme ● Anwendungen, z. B. in der Steuerungstechnik, Medizinelektronik, Konsumelektronik, Meßtechnik ● Praxisnahe Hinweise für Entwickler (µC-Praxis) ● Meß- und Prüftechnik für Mikroprozessorsysteme.

Die ELEKTRONIK informiert Sie über das Produktangebot auf dem internationalen Markt:
● Bauelemente ● Meßgeräte
● Mikrocomputer ● Datentechnik
● Fertigungsmittel ● Software
● Steuer- und Regeltechnik
● Prüftechnik

Die ELEKTRONIK informiert Sie über Aktuelles in der Branche:
Kurzmeldungen aus aller Welt zum Thema ● Technologien ● Verfahren ● Neuheiten ● Unternehmen ● Termine ● Personen

Die ELEKTRONIK informiert Sie mit Sonderpublikationen als Heft im Heft über ● **CAD/CAM** Rechnerunterstützte Entwicklung und Fertigung ● **COM & PRO** Computer und Programme für Anwender von OEM-Produkten ● **TELECOM** Bausteine und Verfahren der Telekommunikation ● **ROBOTER** Flexible Automatisierung in der Industrie

Bitte verwenden Sie zum Kennenlernen unserer ELEKTRONIK nebenstehende Karte.

GARANTIE: **Wenn Sie von unserem KENNENLERN-ANGEBOT Gebrauch machen, können Sie innerhalb von 10 Tagen nach Erhalt des zweiten kostenlosen Heftes das Abonnement widerrufen! Zur Wahrung dieser Frist genügt rechtzeitiges Absenden an den Franzis-Verlag, Zeitschriftenvertrieb, Karlstraße 37, 8000 München 2. Ein Jahresabonnement der ELEKTRONIK kostet DM 126.–, Ausland DM 150.–. Verbilligtes Jahresabonnement für Auszubildende und Studenten: DM 108.–, Ausland DM 132.–.**